AUBURN UNIV ALA ENGINEERING EXPERIMENT STATION REAL-TIME PHASED ARRAY RADAR STUDIES. (U) 1976 E R GRAF, H T NAGLE AD-A032 121 F/G 17/9 DAAH01-71-C-1303 UNCLASSIFIED 10F5 AB32121

(12)

ELECTRICAL

ADA 032121

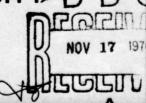
DISTRIBUTION STATEMENT

Approved for public release
Distribution Unlimited

ENGINEERING EXPERIMENT STATION

AUBURN UNIVERSITY DD

AUBURN, ALABAMA



(12)

FINAL TECHNICAL REPORT Contract DAAHO1-71-C-1303

REAL-TIME PHASED ARRAY RADAR STUDIES

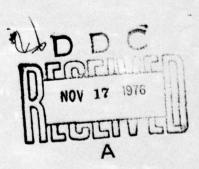
E. R. Graf and H. T. Nagle, Jr. Co-Project Leaders

Prepared for U. S. Army Missile Command Redstone Arsenal, Alabama

Approved for public releases

Distribution Salimited

Prepared by
Electrical Engineering Department
Auburn University
Auburn, Alabama



SECURITY CLASSIFICATION OF THIS PAGE (When Date Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO	3. RECIPIENT'S CATALOG NUMBER
REAL-TIME PHASED ARRAY RADAR STUD	IIII	FINAL TECHNICAL REPORT
	TV C	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(a)		8. CONTRACT OR GRANT NUMBER(*)
E. R. Graf and H. T. Nagle, Jr7	(15	DAAH01-71-C-1303
PERFORMING ORGANIZATION NAME AND ADDRESS Engineering Experiment Station Auburn University Auburn, Alabama 36830		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
U. S. Army Missile Command Redstone Arsenal, Alabama 36809		1976
14. MONITORING AGENCY NAME & ADDRESS(If different	from Controlling Office)	15. SECURITY CLASS OF this report
		UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report)		
Distribution of this report is un		
17. DISTRIBUTION STATEMENT (of the abatract entered in	in Block 20, il different fro	om Report)
18. SUPPLEMENTARY NOTES		
Operating System, Radar Control S Interaction, Module Performance,	oftware, Experin	mental Army Radar, Process
This final report describes System (RADACS) for an SEL86 comp set. The report is composed of 5 of the contract performance.	components of a uter controlled	real-time Radar Operating cross-bow phased array radar
1		402 95

### **FOREWORD**

This final technical report is submitted to the U. S. Army Missile Command by Auburn University to complete its contract obligations under contract DAAHO1-71-C-1303. The report is published in five parts, each separate and independent of the others. Complete computer program listing and card decks have been delivered to the U. S. Army Missile Command under separate cover.



## FINAL REPORT CONTENTS

Part One: RADACS Module Descriptions

Part Two: On the Correctness of a Class of Process Interaction Primitives

Part Three: Module Performance Measurement

Part Four: GTE-Sylvania Color CRT Programmer's Reference Manual

Part Five: An Adaptive Gradient Based Technique for Accelerating the

Convergence Time of Adaptive Array Antennas

## PART ONE

# RADACS MODULE DESCRIPTIONS

# Prepared for

U.S. Army Missile Command Redstone Arsenal, Alabama

Under

Contract DAAHO1-71-C-1303

by

Electrical Engineering Department Auburn University Auburn, Alabama

Prepared by: Joe E. Hawkins, Jr.

Reviewed by: H. Troy Nagle, Jr. Co-Project Leader

# TABLE OF CONTENTS

I.	General	1
II.	Start-Up and Other Initial System Routines	2
III.	System Interrupts	7
IV.	System Services	26
٧.	System Subroutines	59
VI.	System Procedures	154
VII.	System Processes	182
VIII.	Some System Control Blocks	197
Refere	nces	203

## I. GENERAL

This report is an update on system components of the RADACS operating system described in [2]. Only new modules and revised older modules are included in this document. The reader should consider this report to merely be an update of [2].

II. START UP AND OTHER INITIAL
SYSTEM ROUTINES

# INITIAL RADAR OPERATING SYSTEM MODULE (RADACS)

RADACS should always be the first radar operating system module. It has an external reference to all of the radar system modules to see that they are all loaded. The only code in RADACS is a jump to program MASTER.

## SYSTEM START UP ROUTINE (MASTER)

MASTER has control of the machine before the operating system is started (for flowchart see Figure 1). It will accept simple commands for starting the system. MASTER first asks for a command by printing a question on the teletype. Next a line is read from the teletype. If the line is null it is ignored. The first four letters are the command. The command table is searched until the command is found. It will always be found because the input command is put as the last command. When the command is found, MASTER switches to the routine for command processing. An invalid command prints out an error message and returns for another command.

The start command initializes the system and begins execution. A move table in the data base tells what areas in the data base are to be moved elsewhere. A table in the data base indicates certain lists which are to be set up in free memory. The memory tables are set up and all memory used by the system is allocated. The system interupts are enabled as indicated by another table in the data base. Lastly, control is transferred to the program set up as the current process.

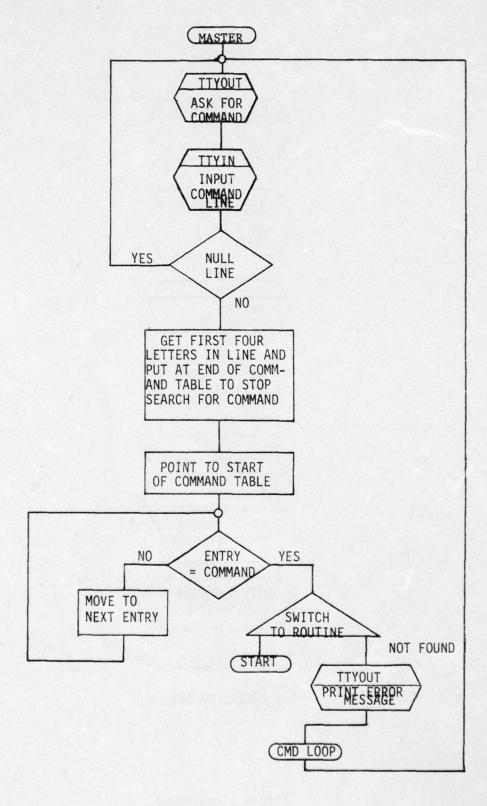


FIGURE 1. FLOWCHART OF MASTER

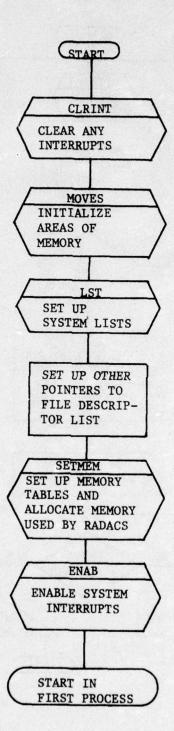


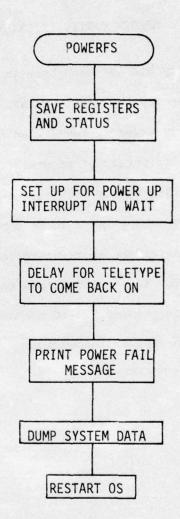
Figure 1 continued

III. SYSTEM INTERRPTS

#### POWER FAIL SAFE INTERRUPT AND TRAP (POWERFS)

The SEL 86 has a power fail safe feature to detect a power failure. The power fail safe feature causes an interrupt or trap to be generated each time the ac power is turned on or off. This interrupt can be used to save the machine status in the non-volatile core memory when power fails and to restart the program when the power comes back on. If the interrupt does not occur in a reasonable time then the trap occurs[1].

The radar operating system handles the interrupt or trap by saving the status and registers when the power fails and by printing a message and restarting the operating system when the power comes on (See figure 2 ). The interrupt and trap dedicated memory locations are set up in NUCINIT. The power fail routine first saves the general registers and the protect registers. Next it saves the old PSW. Last it puts the address of the power on routine in the power fail dedicated memory locations (interrupt and trap) and waits for the power on interrupt. The power on routine delays long enough for the teletype to come back on. Next a power fail message is printed on the teletype including the contents of the registers and PSW when the power failed. After typing the message on the teletype system data is dumped out. After the dump the operating system is restarted.



FIUGRE 2. FLOWCHART OF POWER FAIL ROUTINE (POWERFS)

#### MEMORY PARITY INTERRUPT (MEMPT)

There is a parity bit for each byte in memory. Each time a word is read from memory the parity of all bytes in the word is checked. If a parity error is found when the CPU reads a word then the memory parity interrupt occurs[1].

The radar operating system handles the memory parity error by halting and allowing the operator to do some manual inspection (See figure 3). After the operator restarts the computer then the status information is printed on the teletype and system data is dumped to the line printer. Next the operating system is restarted. The interrupt dedicated memory location is set up and the level enabled in NUCINIT.

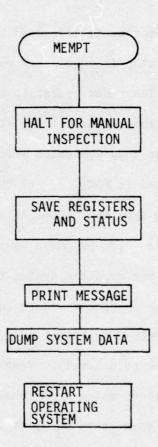


FIGURE 3. FLOWCHART OF MEMORY PARITY ROUTINE (MEMPT)

#### NON-PRESENT MEMORY TRAP (NPMT)

When non-present memory is addressed by the central processor, the instruction execution is terminated and the trap processing routine is entered. The radar operating system takes one of two actions depending on whether or not the NUCLEUS was executing (See figure 4). The trap dedicated memory location is set up and the trap is enabled in NUCINIT. If the trap occured while the NUCLEUS was executing then the system is aborted with an abort message printed on the teletype and a dump of system data on the line printer. After the dump the system tries to restart. If the trap occured while a process was executing then the trap routine enters the NUCLEUS (activate calm level, save registers & status in PCB, clear trap level). Next it aborts the process by placing it on an abort queue and signaling an abort process.

After action has been taken to abort the current process a new current process is selected and control is returned to the new current process.

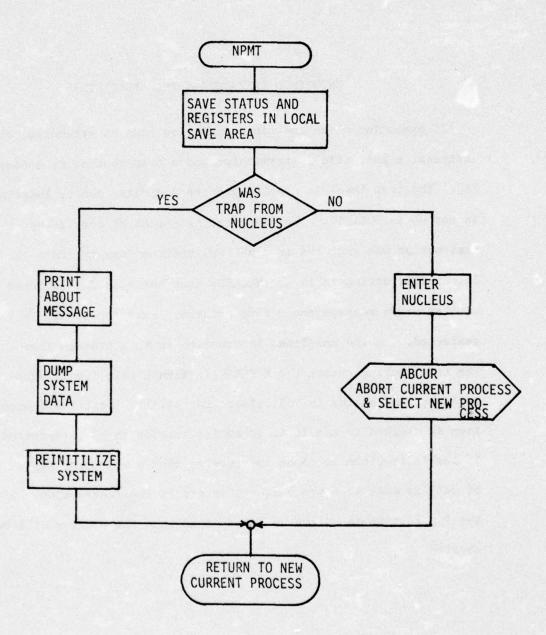


FIGURE 4. FLOWCHART OF NON-PRESENT MEMORY ROUTINE (NPMT)

#### UNDEFINED INSTRUCTION TRAP (UNDINTP)

If execution of an undefined operation code is attempted, the instruction execution is terminated and a trap routine is entered [1]. The trap level is enabled and the dedicated memory location is set up in NUCINIT. The trap routine checks to see if the instruction was executed in a NUCLEUS routine (See figure 5 ). If the instruction is in the NUCLEUS then the system is aborted with an error message and a dump printed. Next the system is restarted. If the undefined instruction is in a process then the trap routine enters the NUCLEUS (activate calm level, save registers and status in PCB, clear trap level). Next the instruction is checked to see if it is an instruction to be interpreted. If the instruction is to be interpreted then a procedure type of call is made to a procedure to interpret the instruction. If the instruction is not to be interpreted then the process will be aborted.

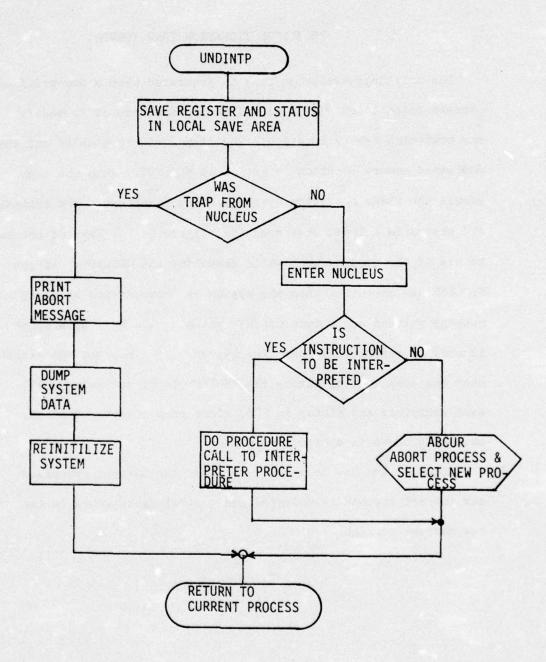


FIGURE 5. FLOWCHART OF UNDEFINED INSTURCTION TRAP ROUTINE (UNDINTP)

#### PRIVILEGE VIOLATION TRAP (PRVT)

The privilege violation trap is generated when a non-privileged process tries to execute a privileged instruction or to modify any protected memory location. The trap level is enabled and the dedicated memory location is set up in NUCINIT. When the trap occurs the radar operating system routine first saves the registers and status in a local save area (See figure 6 ). Next it checks to see if the trap occured while executing the NUCLEUS. If the NUCLEUS was executing then the system is aborted with an abort message printed and a dump of the system data. Next an attempt is made to restart the operating system. If a process was executing then the trap routine enters the NUCLEUS (activate calm level, save registers and status in PCB, clear trap level). Next the current process is aborted.

After action has been taken to abort the current process a new current process is selected and control is returned to the new current process.

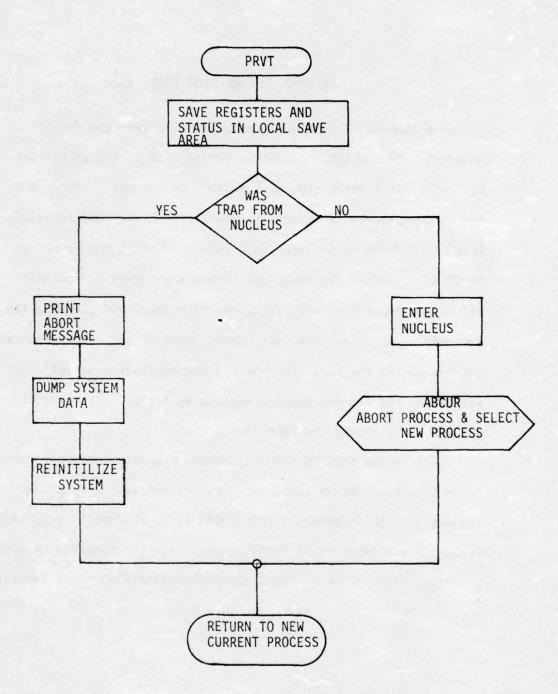


FIGURE 6: FLOWCHART OF PRIVILEGE VIOLATION TRAP (PRVT)

#### ARITHMETIC EXCEPTION (ARTHEXCP)

An Arithmetic Exception interrupt occurs when the results of an arithmetic operation cannot be represented in the machine[1]. The radar operating system handles this by setting a bit in the PSW of the process it occurs in (See figure 7 ). The interrupt level is enabled in the dedicated memory location and is set up in NUCINIT. After the interrupt occurs the interrupt routine enters the NUCLEUS by activating the CALM level and clearing the arithmetic exception interrupt level. Next it saves the registers and the PSW in the PCB. The bit for the arithmetic exception is set in the PCB and the routine returns by restoring the registers and PSW and clearing the CALM level.

While in the NUCLEUS the arithmetic exception interrupt cannot occur because a higher level interrupt is active. However, a request for the interrupt is generated if an arithmetic exception occurs. To prevent an interrupt when control is returned to a process the arithmetic exception level should be cleared whenever leaving the nucleus.

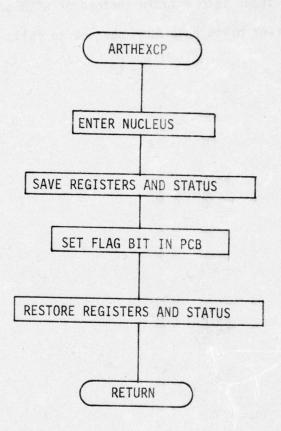


FIGURE 7. FLOWCHART OF ARITHMETIC EXCEPTION ROUTINE (ARTHEXCP)

# SERVICE INTERRUPT MAPPED (SIMAP)

SIMAP has no major changes. The SI semaphore pointers are now found by an index into a table instead of with global capability numbers. Error halts have been changed to calls on SYSABORT.

# REAL TIME CLOCK INTERRUPT (RTCINT)

RTCINT has no major changes to it. Error halts have been replaced with calls to SYSABORT. Code to access data in PCB's has been changed because of changes in the PCB format.

## CALM MONITOR INTERRUPT HANDLER (MONITOR)

MONITOR has had two sections of code added and the services table updated. A section was added to handle calls to supervisor procdures. Also a section was added to the monitor exit protion to set up the memory protect registers and to clear the arithmetic exception interrupt in the event it was set while in the monitor. The updated flowchart is in figure 8.

The supervisor call section is entered when the CALM code is too large for a NUCLUS service. The code is checked to see if it is in the supervisor range, if not then it is treated as an error. If the code is in range it is used as an index into a table to get the capability for the supervisor procedure. If there is no capability it is treated the same as when the code is out of range. The capability is used to do a procedure call. Upon exit from MONITOR the procedure is in control.

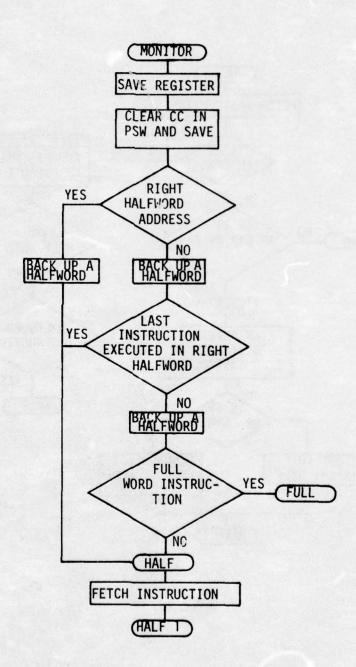


Figure 8 . Flowchart of Monitor

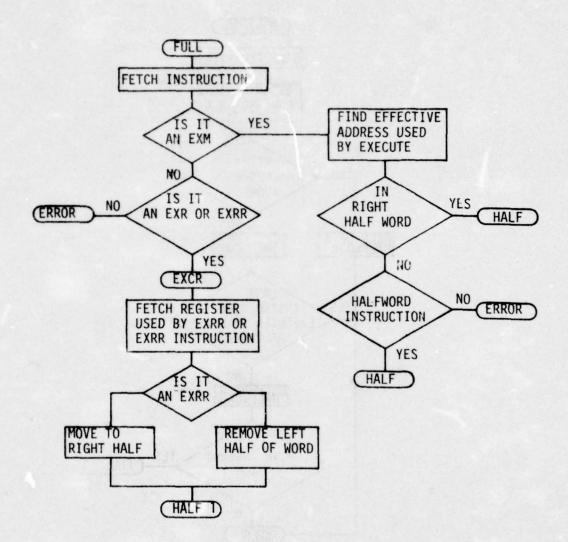


FIGURE 8 . Continued

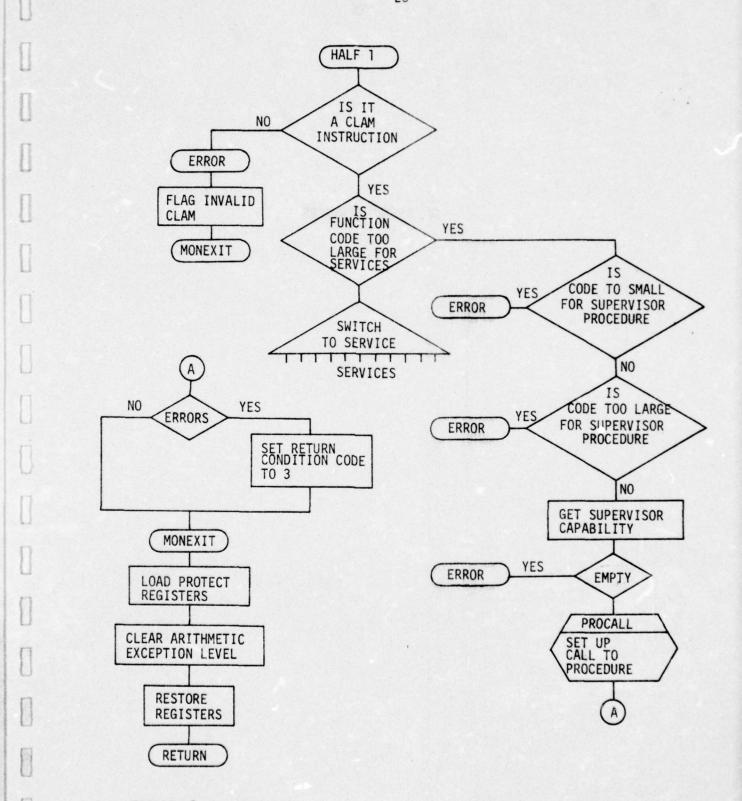


FIGURE 8 . CONTINUED

... IV. SYSTEM SERVICES

#### ALLOCATE CAPABILITY SERVICE ROUTINE (ALLOCATE)

ALLOCATE creates a capability of the type requested by the current process from information in the process's registers. A pointer to the capability descriptor is put in the process's capability list (C-list).

ALLOCATE receives control from MONITOR when a process requests that a capability be created (see Figure 9 ). First the capability type is checked to see if it is valid. If it is not then ALLOCATE sets the condition code and returns through MONEXIT. Next, ALLOCATE calls a routine to find a slot in the C-list (see Figure 10 ). If none is found the condition code is set and ALLOCATE returns through MONEXIT. Next ALLOCATE gets the type of block needed for the capability type if one is needed. If a block cannot be found the condition code is set and ALLOCATE returns through MONEXIT. ALLOCATE then switches to the code for processing the requested capability type. The capability types are EVENT, MESSAGE, PROCEDURE, DYNAMIC MEMORY, and SON PROCESSES.

After doing the individual processing for each of the capability types ALLOCATE completes the processing in a common section of code. The capability name is put in the small block used to describe the capability. Next a pointer to the capability is put into the empty place in the C-list. Lastly, ALLOCATE returns through MONEXIT.

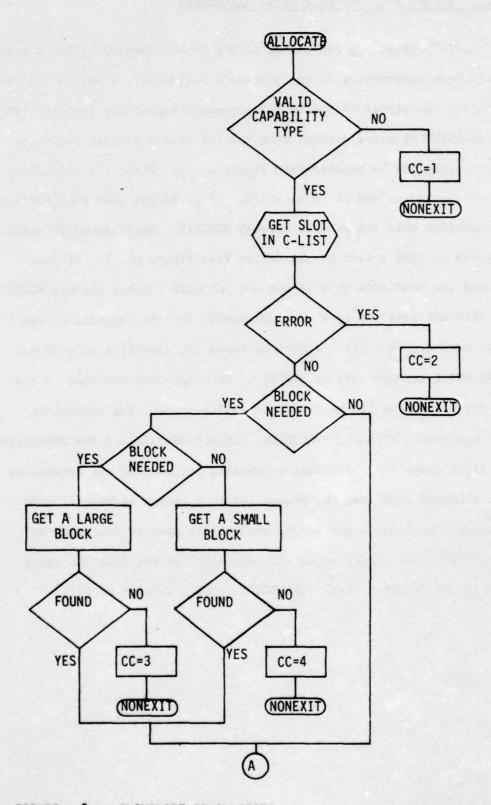


FIGURE 9 . FLOWCHART OF ALLOCATE.

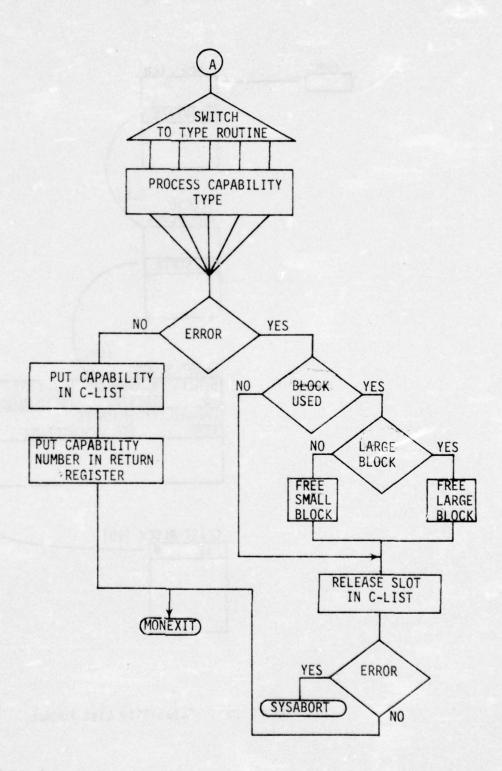


FIGURE 9 . CONTINUED

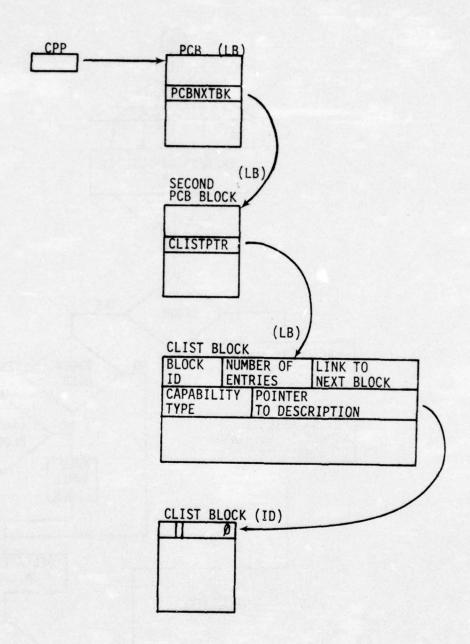


Figure 10 . Capability List Format

# ATTACH CAPABILITY SERVICE ROUTINE (ATTACH)

ATTACH gives access to the capability of the type and name indicated by the process. The capability lists of the processes ancestors are searched for the capability with the desired name and type. When found, a pointer to the capability is put in the current process's C-list.

ATTACH receives control from MONITOR when a process requests the attach service (see Figure 11 ). First the capability type is checked to see if it is valid. If it is not valid then ATTACH sets the condition code and returns through MONEXIT. Next ATTACH searches the capability lists for the named capability. If it is not found then ATTACH sets the condition code and returns through MONEXIT. The process's C-list is searched for an empty capability pointer. If none is found then ATTACH sets the condition code and returns through MONEXIT. Next any data the process needs out of the capability descriptor is copied into the process's registers. The capability-use counter is incremented by one to indicate a new user. Next a pointer to the capability is put in the current process's C-list. Lastly, ATTACH returns theough MONEXIT.

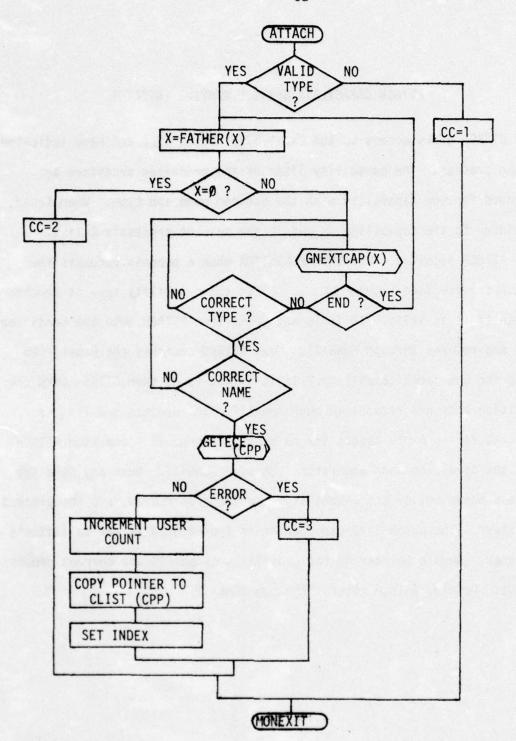


FIGURE 11 . FLOWCHART OF ATTACH

DETACH CAPABILITY SERVICE ROUTINE (DETACH)

DETACH removes access to the indicated capability from the current process. The capability number supplied by the process is used as an index into the process's c-list. The user count of the capability pointed to by the c-list pointer is decremented by one. If the user count goes to zero then the capability is removed from the system. Also the pointer in the c-list is cleared.

DETACH receives control from MONITOR when a process requests the detach service (See figure 12 ). FINDCAP is called to get the pointer to the capability. If the capability is not found then DETACH sets the process's condition code and returns through MONEXIT. If found RMVCAP is called to remove the capability.

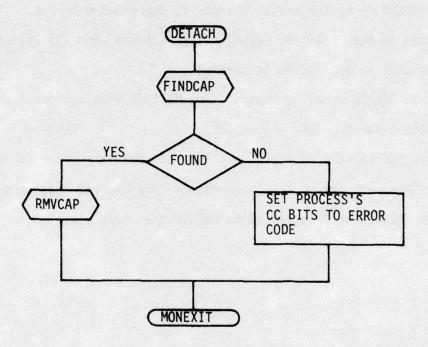


FIGURE 12. FLOWCHART OF DETACH

### CALL PROCEDURE SERVICE ROUTINE (CALL)

CALL passes control to a procedure for which the process has a capability (See figure 13). GETCAP is called to find the capability descriptor. If it is not found then CALL returns with an error code. The capability type is checked to see that it is a procedure. PROCCALL is called to set up for the call by putting return data on the return stack and making changes in the Process Control Block. If PROCCALL could not set up for the call properly then CALL returns with an error code.

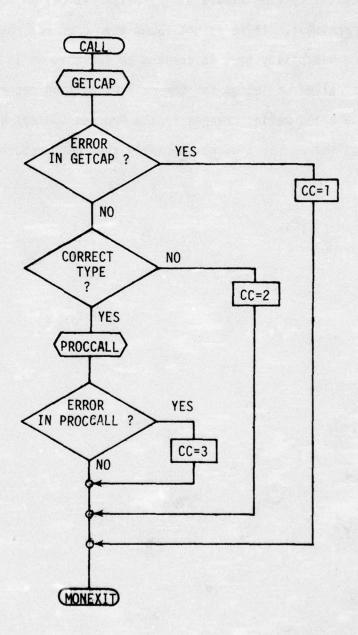


Figure 13, Flowchart of CALL

# RETURN FROM PROCEDURE SERVICE ROUTINE (RETURN)

RETURN restores a PCB for a return from a procedure (See Figure 14). If there is no return block on the return stack, indicating this is a main procedure, then the terminate service is entered to remove the process from the system. The data in the return block is copied back into the PCBC (registers, PSW, and old procedure protect bits). The procedure and memory protect bits are combined to produce the protect bits actually used. Last the large block used for the return block is removed from the return stack and freed.

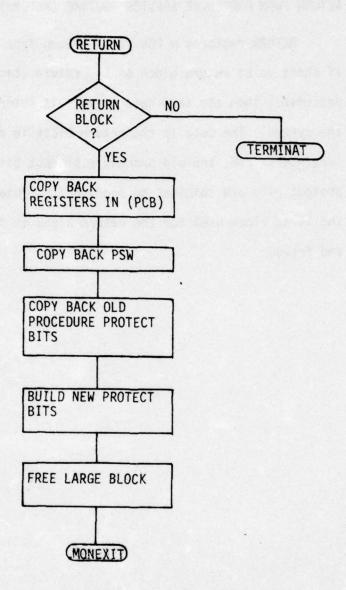


Figure 14 FLOWCHART OF RETURN

# TERMINATE CURRENT PROCESS SERVICE ROUTINE (TERMINAT)

TERMINAT stops execution of the current process and its descendents (see Figure 15). If the current process is essential to system operation then the system will be aborted by SYSABORT. If the current process does not have a father with a c-list entry for the current process then some system data must be corrupted, so the system is aborted by SYSABORT. The c-list entry of the father for the current process is changed to a terminated son type. The terminate code is put in the current process's PCB and the process is put on the terminate queue. A new current process is selected by SELNCUR. If the father process has a queue for terminate messages PROSEND is used to send a message to that queue. FLGDSCND is called to flag all descendents of the terminating process with a terminate code, so they will no longer execute. VOP is used to signal the termination process that a process is on the terminate queue. If VOP fails then some system data must be corrupt, so the system is aborted by SYSABORT.

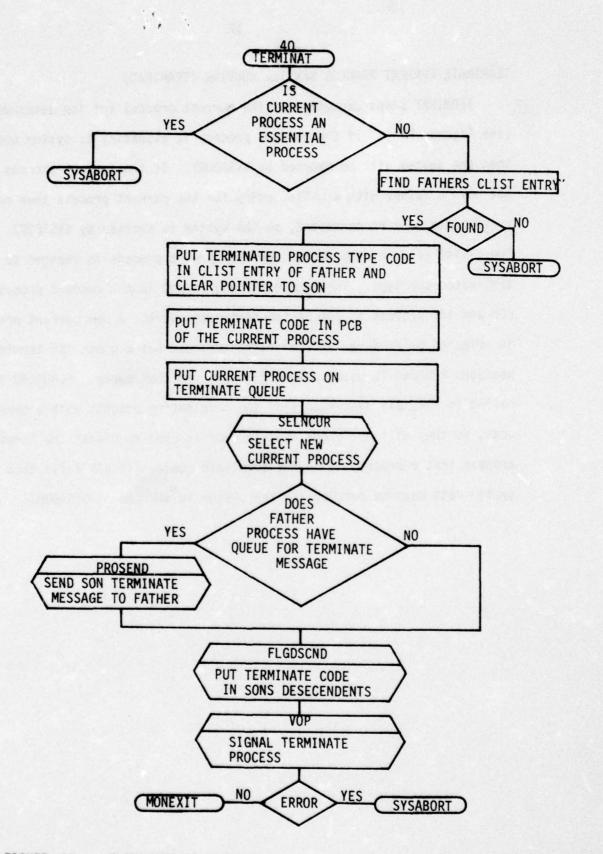


FIGURE 15 FLOWCHART OF TERMINAT

# RETURN REGISTER SERVICE ROUTINE (RTNREG)

RTNREG puts a return value in the saved register area on the procedure return stack (see Figure 16 ). The number indicating the source register is checked to see if it is in range. If it is not then RTNREG returns with an error code in the condition code. The number indicating the destination register in the save area is checked to see that it is in range. If it is not in range then RTNREG returns with an error code.

Next the return stack is checked to see if there is a return block. If no return block then RTNREG returns with an error code. Last the value in the source register is copied into the return block.

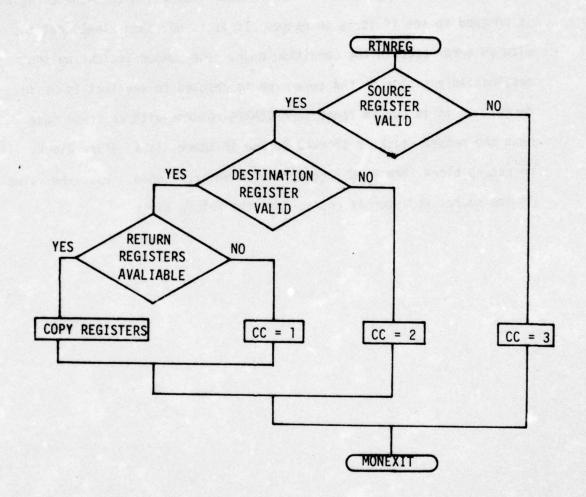


FIGURE 16. FLOWCHART OF RTNREG

### SET CONDITION CODE SERVICE ROUTINE (SETCC)

SETCC sets the cc bits in the PSW in the procedure return block and returns the previous value of the bits (see Figure 17). SETCC first checks to see that a return block is available. If not then an error code is returned. The old value of the condition code bits in the return block is saved. The new value is placed in the return PSW in the procedure return block. Last the old condition code value is returned in the current process's register.

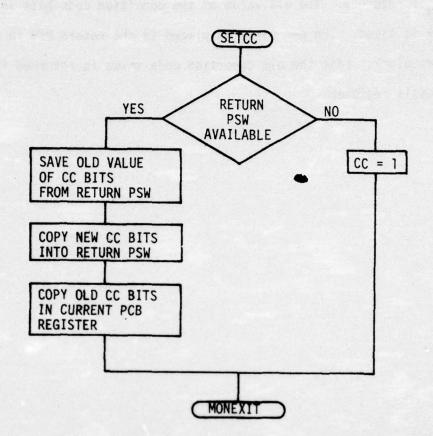


FIGURE 17. FLOWCHART OF SETCC

#### FILE DESCRIPTOR

The file descriptor is used to describe open files. The descriptors are in a table and free descriptors are on a linked list. Figure 18 shows the layout of the descriptor. A description of each entry follows.

- Word Ø When the descritpor is free the last 20 bits point to the next free block. When in use the last 20 bits point to the PCB of the process that opened the file.
- Word 1 When used as an operator input file contains the name or character the operator uses to access it, should be a zero at other times.
- Word 2 Counter of the semaphore the process will block on while waiting for the I/O to finish.
- Word 3 Queue pointer for semaphore to blocked process.
- Word 4 Transfer control word verified to be in memory accesable to the process.
- Word 5 System capability for memory containing the I/O buffer.
- Word 6 System capability for memory containing the File Control Block.
- Word 7 Status Flags and pointer to the File Control Block.

  Bit Ø Busy Flag

  Bit 1 Blocked for close

  Bit 2 Message waiting form Operator input

  Bit 3 File waiting for operator input

  Bits 12-31 Pointer to File Control Block
- Word 8 System capability for device semaphore (used to send the start message to the device driver process).
- Word 9 Pointer to disc partition data of file.
- Word 10 Pointer to buffer holding a message from the operator.
- Word 11 Not used.

1B	4 BITS	20 BITS
		POINTER TO PCB
	OPCOM I	NPUT NAME
	co	DUNTER
	Q	PTR
		TCW .
	SYSTEM CAP	. FOR I/O BUF
	SYSTEM C	AP. FOR FCB
FLAGS		POINTER TO FCB
750 2015	CAP. FOR	DEV. SEM.
		POINTER TO PART. DATA
		POINTER TO WAITING MSG

BIT Ø BUSY FLAG

BLOCKED FOR CLOSE

2 MESSAGE WAITING FOR OP INPUT 3 WAITING FOR OPCOM MSG

FIGURE 18 . FILE DESCRIPTOR

OPEN FILE SERVICE ROUTINE (OPEN)

OPEN sets up a file descriptor for a file. It attaches the descriptor to the I/O process semaphore, the calling processes FCB and to a disc partition if a disc file. For the relation between I/O system data see Figure 19 . For the flowchart of OPEN see Figure 20.

FINDCAP is called to find the capability for the block of memory containing the FCB. If the capability is not found an error code of one is returned to the calling process. The FCB pointer is checked to see that the FCB is within the memory block. If it is not then an error code of two is returned to the calling process. OPEN searches the File Assignment Table for the file indicated in the FCB. If it is not found then an error code of 3 is returned to the calling process. A file descriptor is removed from the Free FILE Descriptor list. If none are available then an error code of four is returned to the calling process. The pointer to the device semaphore and partition table are copied into the file descriptor. The pointer to the FCB memory block descriptor is put in and the user count in the memory block descriptor is incremented. A pointer to the calling process's PCB is put in the descriptor. The operator communications file name is copied in and other data locations are cleared in the file descriptor. A pointer into the file descriptor table is put in the calling process's FCB and OPEN returns through MONEXIT.

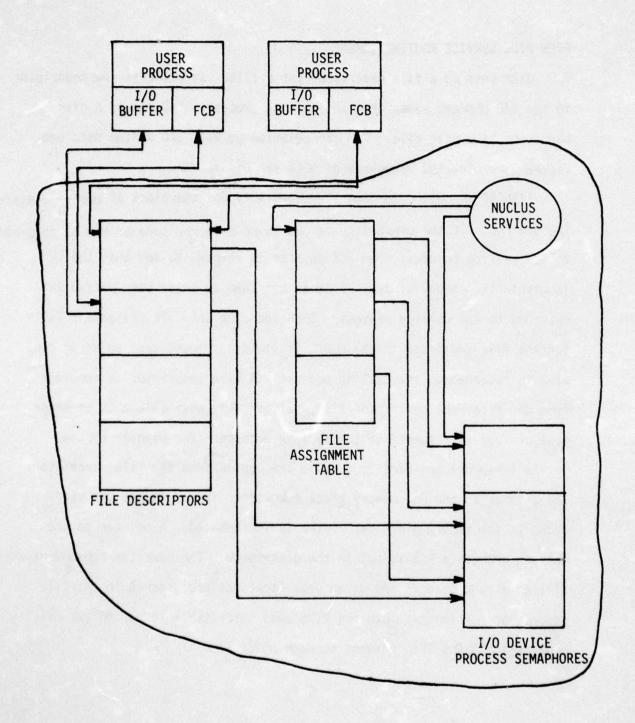


FIGURE 19. I/O SYSTEM DATA RELATIONS

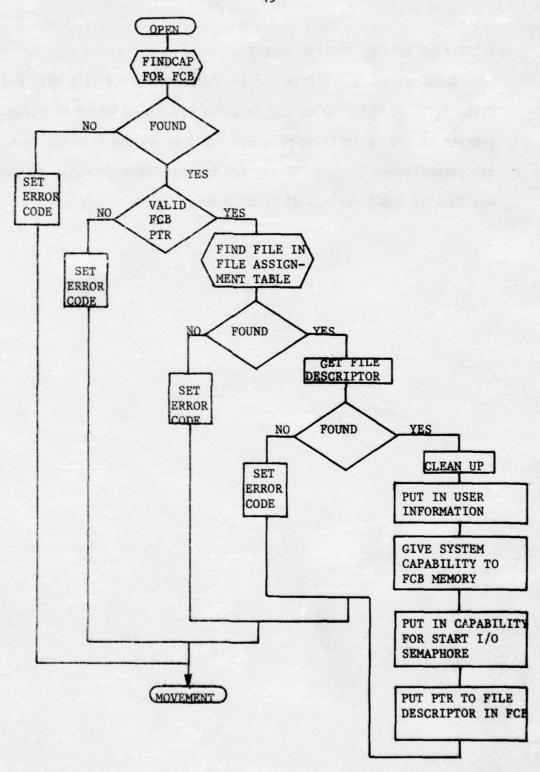


Figure 20. FLOW CHART OF OPEN

# CLOSE FILE SERVICE ROUTINE (CLOSE)

CLOSE closes out a file for the current process (see figure 21).

CLOSE calls IO.CLOSE to do the actual close operation of blocking the process if I/O is still in progress and freeing the file descriptor and capabilities for the FCB and I/O buffer memory if not. If the FCB was invalid CLOSE returns an error code.

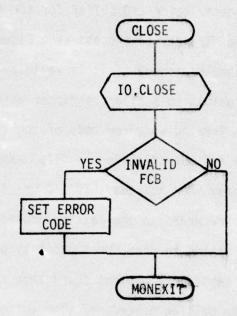


FIGURE 21. FLOWCHART OF CLOSE

### START I/O SERVICE ROUTINE (STARTIO)

STARTIO secures access to the I/O buffer for the system and sends the start command to the I/O device process (see Figure 22 ). The pointer to the FCB is checked to see if it is valid by checking to see if it has a pointer which points to a file descriptor which points back to the FCB. If the FCB is invalid an error code of one is returned to the calling procedure. The I/O busy flag in the file descriptor is checked to see if any I/O has already been started. If I/O was already started then an error code of two is returned to the calling procedure. If no I/O was started FINDCAP is called to find the capability pointer for the I/O buffer memory. If the capability was not found then an error code of three is returned to the calling procedure. The buffer memory address and size is checked to see that it is in the block described by the capability descriptor. If the buffer is not in the block then an error code of four is returned to the calling procedure. The system is given a capability to use the I/O buffer memory by putting a pointer to the capability descriptor in the file descriptor and incrementing the user count in the capability descriptor. The Transfer Control Word from the FCB is copied into the file descriptor. A start message is sent to the I/O process pointed to in the file descriptor. If the message could not be sent an error message of five is returned to the calling procedure. After all processing is done STARTIO returns to the calling process through MONEXIT.

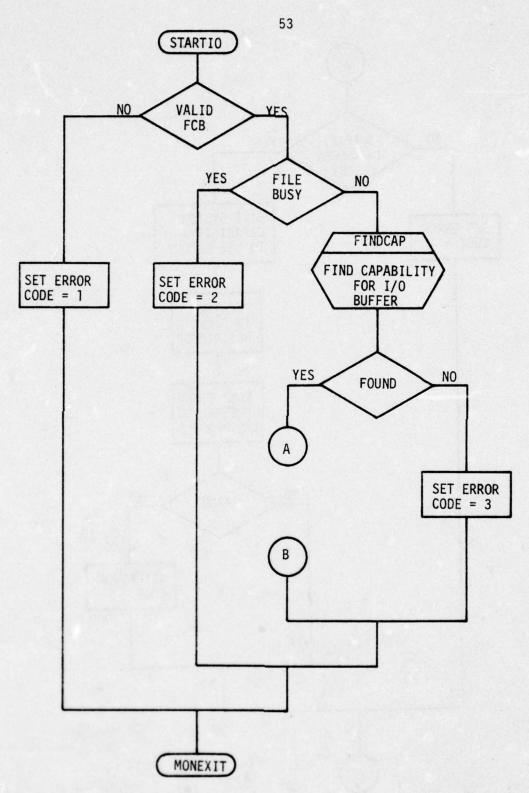


Figure 22. Flowchart of STARTIO

Figure 22 continued

WAIT FOR I/O SERVICE ROUTINE (WAITIO)

WAITIO checks for completion on an input/output operation and waits for completion if requested (see Figure 23 ). The pointer to the FCB is checked to see if it is valid by checking to see if it has a pointer which points to a file descriptor which points to it. If the FCB is invalid an error code of one is returned to the calling procedure. The I/O busy flag in the file descriptor is checked to see if any I/O was started. If no I/O was started then an error code of two is returned to the calling process. The semaphore counter in the file descriptor is checked to see if it is greater than zero, indicating that the I/O operation is finished. If the I/O is still in progress WAITIO checks the maximum wait time to see if any waiting is wanted. If the wait time is negative, indicating no waiting, then the error code of four is returned to the calling process to indicate that the I/O is still busy. If waiting is desired then POP is called to block the calling process on the semaphore in the file descriptor. If POP detects any errors then an error code of three is returned to the calling process. If the I/O is finished then WAITIO completes the I/O operation. The semaphore counter in the file descriptor is decremented. The I/O busy flag is cleared. The pointer to the memory descriptor for the buffer memory is cleared and the user count in the memory descriptor is decremented. If the user count goes to zero then RV.MEM is called to free the buffer memory. Last control is returned to the calling process through MONEXIT.

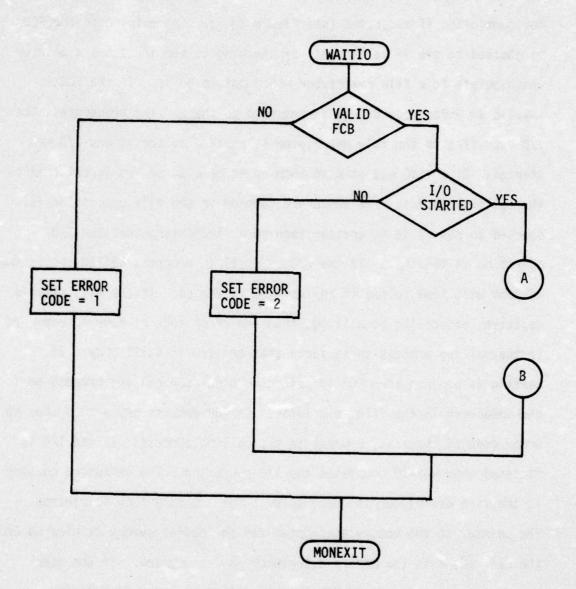


Figure 23 Flowchart of WAITIO

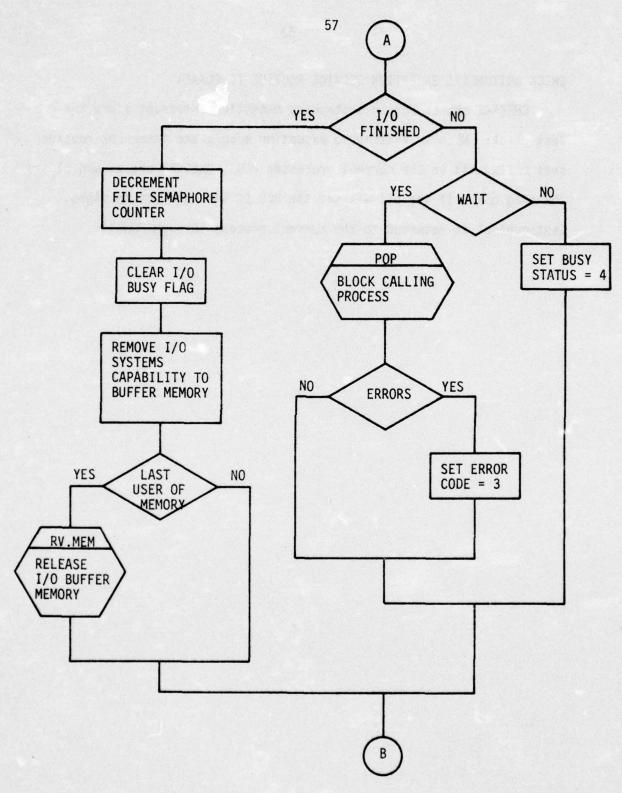


Figure 23. Continued

# CHECK ARITHMETIC EXCEPTION SERVICE ROUTINE (CHECKAE)

CHECKAE checks for an arithmetic exception interrupt since the last check. When an arithmetic exception occurs the interrupt routine sets a flag bit in the current processes PCB. CHECKAE checks and clears the flag bit. If the bit was set the PCB CC bits are set to eight. Last control is returned to the current process through MONEXIT.

V. SYSTEM SUBROUTINES

## GET CAPABILITY POINTER SERVICE SUBROUTINE (GETCAP)

GETCAP is called by some of the service routines to get a pointer to the capability descriptor and the capability type (See figure 24 ). FINDCAP is called to get a pointer to the c-list entry. If the entry was not found GETCAP returns with the condition code set. If found a pointer to the c-list entry, the object type, and a pointer to the descriptor are extracted and returned.

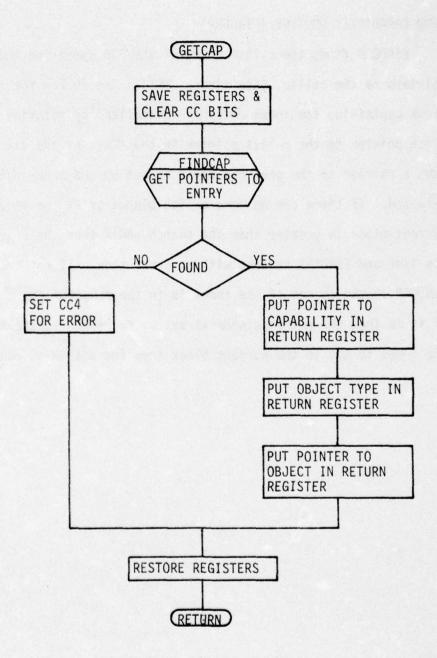


FIGURE 24 FLOWCHART OF GETCAP

FIND CAPABILITY ROUTINE (FINDCAP)

FINDCAP finds the c-list entry of the PCB specified and returns pointers to the caller. (See figure 25 ). The search for the c-list block containing the index entry is initialized by pointing the current block pointer to the c-list pointer in the PCB. At the start of the search loop a pointer to the previous block is set up and a new current block is selected. If there are no more c-list blocks or if the index for the current block is greater than the search index then the entry is not in the list and FINDCAP returns with an error code. If not the end then FINDCAP checks to see if the index is in the range of the current block. If it is then the entry pointer is set up to return to the caller. If the index is not in the current block then the search is continued.

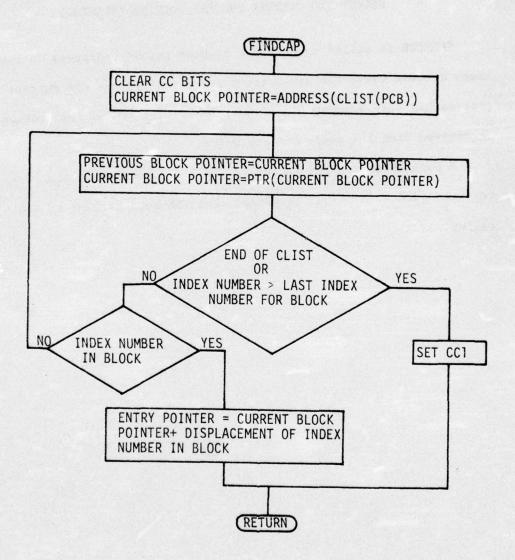


FIGURE 25 FLOWCHART OF FINDCAP

# SELECT NEW CURRENT PROCESS ROUTINE (SELNCUR)

SELNCUR is called to make the highest priority process on the Ready Process Queue the new current process after the old current process has been removed (See figure 26 ). The PCB of the process is removed from the Ready Process Queue.

A pointer to the PCB is put in CPP and a pointer to the register save area is put in CPSAP. Last SELNCUR returns to the caller.

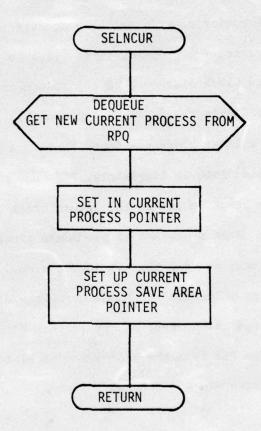


FIGURE 26. FLOWCHART OF SELNCUR

## PROCEDURE CALL SERVICE SUBROUTINE (PROCCALL)

PROCCALL is called to do the data saving and data set up needed for a procedure call (see figure 27 ). A pointer to the block of information describing the procedure to be called is passed in a register. PROCCALL first tries to get a large block from the large block list to save the return information. If no blocks are available then a condition code bit is set and PROCCALL returns. If a large block was found then data needed for a return is copied into it (registers, PSW, and procedure protect bits) and the block is put on the return stack as a procedure return block. Next a new set of procedure protect bits are built using the current set and the set in the procedure descriptor.

A new set of protect bits are built using the new procedure protect bits and the current memory protect bits. Last PROCCALL copies into the PCB from the procedure descriptor the starting PSW for the procedure and returns.

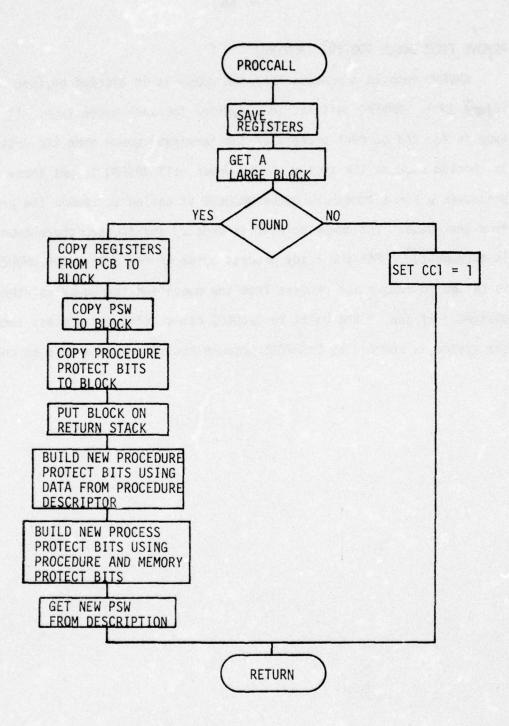


Figure 27 . Flow chart of PROCCALL

## REMOVE FROM QUEUE ROUTINE (RMVFMQ)

RMVFMQ removes a process from the queue it is blocked on (see Figure 28). RMVFMQ switches to a routine for each queue type. If the type is for the current process or the terminate queue then the system is aborted because the system should never call RMVFMQ to get these processes. For a semaphore queue UNQUEUE is called to remove the process from the queue. The queue pointer is updated and the semaphore counter is incremented. For the ready process queue or the hold queue UNQUEUE is called to remove the process from the queue and the queue pointer is updated. If one of the calls to UNQUEUE cannot find the process then the system is aborted by SYSABORT because the system data must be corrupted.

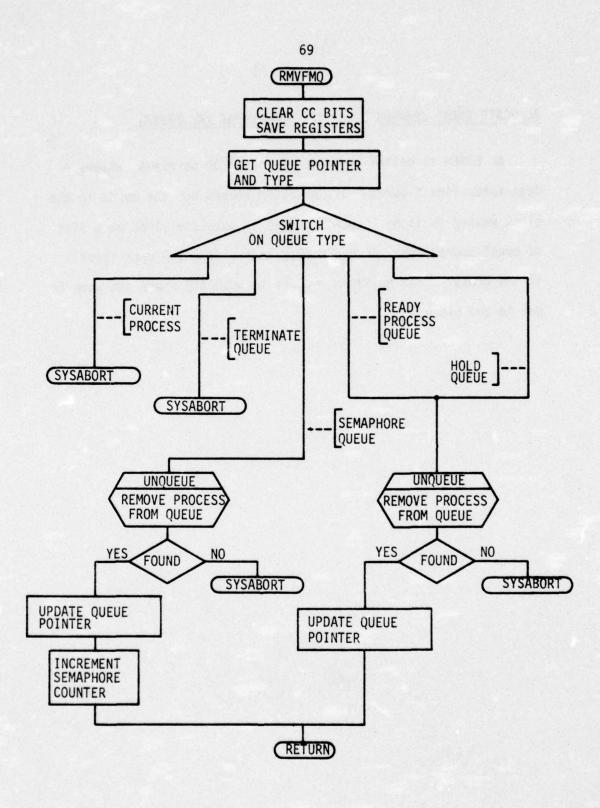


FIGURE 28. FLOWCHART OF RMVFMQ

## ALLOCATE EVENT SEMAPHORE SERVICE SUBROUTINE (AL.EVSEM)

AL.EUSEM is called by ALLOCATE to set up on event semaphore descriptor (see figure 29 ). AL.EVSEM zeroes out the words in the block passed to it by ALLOCATE. Next it puts the block on a list of event semaphores. AL.EVSEM puts in the name and user count in the block. Last AL.EVSEM returns to ALLOCATE where the name is put in the block.

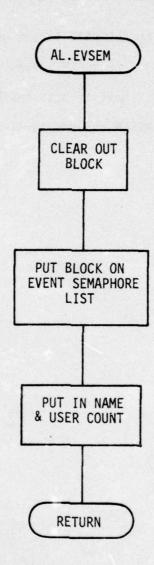


Figure 29. Flowchart of AL.EVSEM

# ALLOCATE MESSAGE SEMAPHORE SERVICE SUBROUTINE (AL.MSSEM)

AL.MSSEM is called by ALLOCATE to set up a message semaphore descriptor (see figure30). AL.MSSEM zeroes out the words in the block passed to it by ALLOCATE. Next it puts the block on a list of message semaphores. AL.MSSEM returns to ALLOCATE where the name is put in the block.

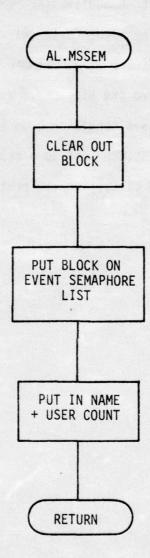


Figure 30. Flowchart of AL.MSSEM

ALLOCATE MEMORY SERVICE SUBROUTINE (AL.MEM)

AL.MEN is called by ALLOCATE to get a block of memory and allocate it to a process (See figure 31). FINDMEM is called to find the block of memory and to reserve the block. If memory could not be found then AL.MEM sets the processes condition code and returns to ALLOCATE. If memory was found then AL.MEM sets up a capability for the memory, gives it to the process, and calculates the protect bits to permit access to the memory.

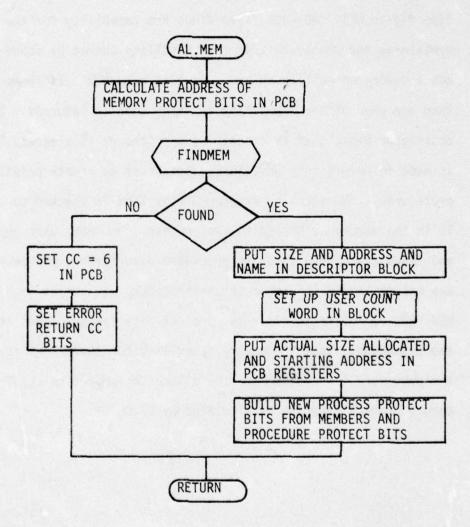


FIGURE 31. FLOWCHART OF AL.MEM

#### AL.PROC SERVICE SUBROUTINE

AL.PROC allocates a procedure by setting up a procedure descriptor (see figure 32). AL.PROC first finds the capability for the memory containing the procedure. If the capability cannot be found or it is not a memory capability an error code is returned. If there is more than one user of the capability an error code is returned. The large descriptor block used to describe the procedure is cleared. Next a check is made to permit only priviledged processes to create priviledged procedures. The starting Program Status Word is checked to see that it is in the memory allocated to the process. The PSW, user count, name, and flags are copied into the procedure discriptor. All memory protect bits are set in the descriptor. If the procedure will write into itself then MEM.PROT is called to clear the protect bits for the pages containing the procedure. Last the memory capability for the memory containing the procedure is removed from the allocating processes CLIST and its memory protect bits are recalculated by PB.CLIST.

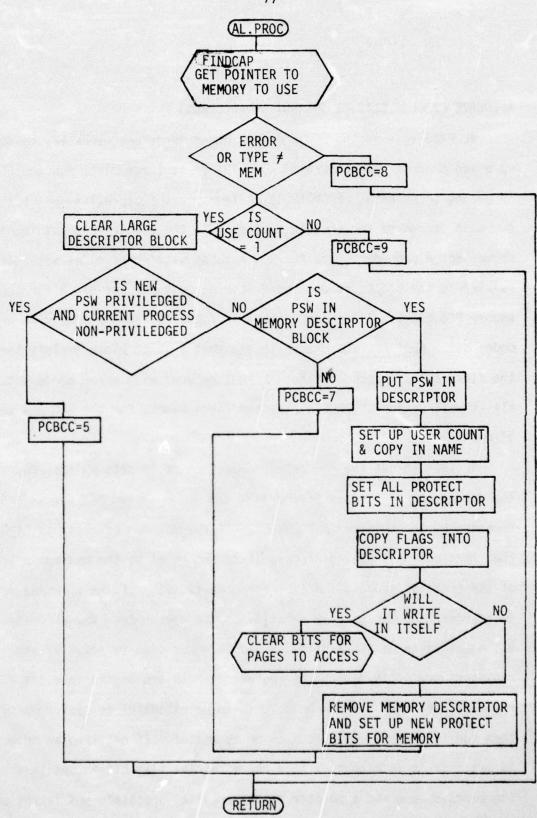


FIGURE 32. FLOWCHART OF AL. PROC

ALLOCATE PROCESS SERVICE SUBROUTINE (AL.PROS)

AL.PROS sets up the PCB and does other functions necessary to start up a son process (see Figure 33 ). The first large block for the PCB is passed to AL.PROS. FINDCAP is called to get a capability pointer to the main procedure for the new process. If the pointer was not found or was not a procedure type AL.PROS returns with error codes set. If the pointer to the procedure was found AL.PROS gets a large block for the second PCB block. If no large block is found AL.PROS returns with error codes set. Next AL.PROS checks to see that a large block is left for the first c-list block. If not AL.PROS returns with error codes set. If all is well so far AL.PROS clears the large blocks for the PCB and sets it up.

AL.PROS checks the requested parameters as it sets up the PCB. For the format of the PCB see Figure's 34 and 35. The two PCB blocks are lined together by the pointer in PCBNXTBK. If the maximum priority is higher than that of the creating process it is set equal to the maximum priority of the creating process and an error code is set. If the starting priority is greater than the maximum priority of the new process the priority is set equal to the maximum priority and an error code is set. If the initial register area is in memory the register values are copied into the PCB register area. If the new process is to be essential to system operation then the father process must also be essential. If not then an error code is set. If it is essential then the essential flag bit is set in the PCB. The process name and a pointer to the creating process's PCB is set up in the new PCB. A large block is gotten for the c-list to put in the capability for the main procedure. The initial PSW and protect bits are set

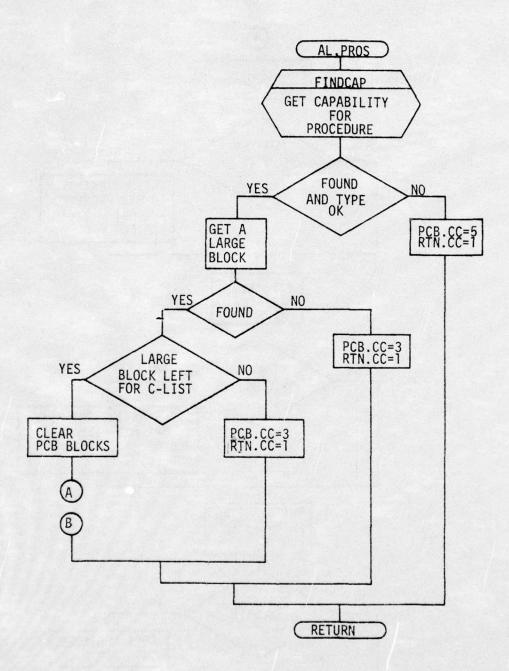


FIGURE 33 . FLOWCHART OF AL.PROS

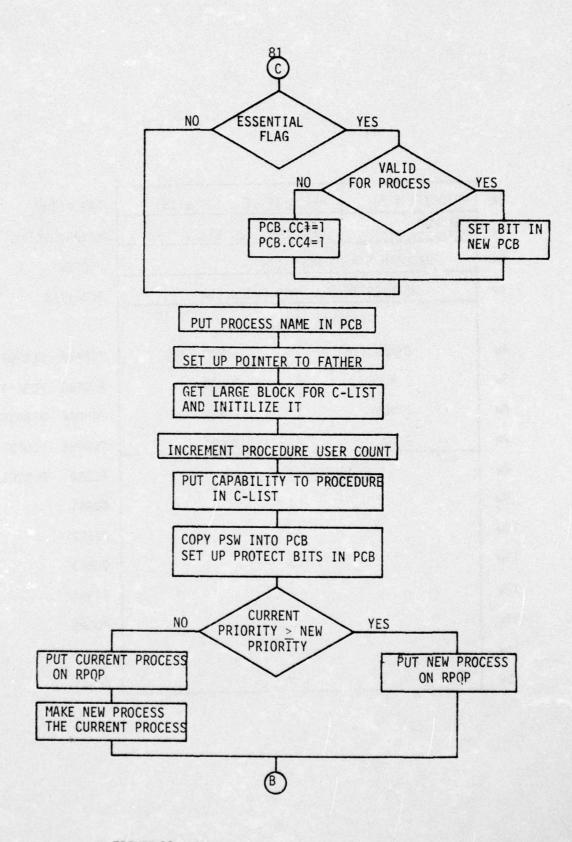


FIGURE 33 CONTINUED

Ow	PRIORITY (1B) PRIO		RITY LINK (20 BITS)		PRI PLI	PRI PLINK		
1w	MAXIMUM PRIORITY (1B)	TIME	LINK (20 BITS)		MAXPRI	MAXPRI TLINK		
2w	PROGRAM STAT	PROGRAM STATUS WORD (1w)				PCBPSW		
3w	POINTER TO	PCB BLOCK	(1w)	PCBNXTE	PCBNXTBK			
	PRO	TECT	BITS					
4w	0 MOD (1H	)	1	MOD (1H)	PCBPBØ	PCBPB1	PCBPBITS	
5w	2 MOD		3	MOD	PCBPB2	PCBPB3		
6w -	4 MOD		5	MOD	PCBPB4	PCBPB5		
7w	6 MOD		7	MOD	PCBPB6	PCBPB7		
8w	REG	ISTER	Ø		PCBRØ	PCBREGS	٧	
9w			1		PCBR1			
10w			2		PCBR2			
11w			3		PCBR3			
12w			4		PCBR4			
13w			5		PCBR5			
14w			6	,	PCBR6			
15w			7		PCBR7			

FIGURE 34 . FIRST PROCESS CONTROL BLOCK

Ow	RETURN LIST POINTER	PCBRLP			
1w	CURRENT QUEUE POINTER	PCBQP			
2w	TIMER CHAIN POINTER	PCBTCP			
3w	C-LIST POINTER	CLISTPTR			
4w	FATHER POINTER (1w)	FATPTR			
5w	FLAGS (1B) TERM CODE (1B) SON TERMINATE MSG Q INDEX (1H) FLGDISP/CODEBYTE/QUEUE				
6w	WORD USED BY GET DESCENDENT	CURCAPEN			
7w	PROCESS NAME (1w)	PCBNAME			
8w	PROTECT BITS FOR	РСВРМØ	PCBPM1	РСВРМВ	
9w	PROCESS MEMORY	2	3		
10w		4	5		
11w		6	7		
12w	PROTECT BITS FOR	РСВРРØ	PCBPP1	РСВРРРВ	
13w	CURRENT ENTERED PROCEDURES	2	3		
14w		4	5		
15w		6	7		

FIGURE 35 . SECOND PROCESS CONTROL BLOCK

up with the information in the procedure's descriptor. If the creating process has a lower priority than the new process then it is put on the ready queue and the new process is made current. If not then the new process is put on the ready queue.

FIND MEMORY SERVICE SUBROUTINE (FINDMEM)

FINDMEM finds available memory and allocates it (See Figure 36 ). FINDMEM calls one of three search and allocation routines each of which searches for a different arrangement of allocated memory. MEM.A tries to find free memory in a page or part of pages already accessible to the process. MEM.B tries to find an accessible page that can be extended to a free page. MEM.C tries to find space starting on a new page and using only new pages.

FINDMEM tries to allocate memory in a way that minimizes fragmentation and maximizes protection between processes. MEM.C is called first only when a new page is requested. MEM.B is called first when the requested size is too large to fit in a pair of accessible pages.

Complete write protection between processes is not guaranteed due to the ability to share blocks of memory and the memory allocation in pages accessible to a process. Processes that do not share memory will never have access to the same page so complete protection is possible for these processes.

Memory usage is kept track of with a bit map. Each bit covers a 16 word block of memory, with a zero bit indicating a free block. Each word in the bit msp represents a 512 word page. The bit map is 64 words long, long enough for 32 K of memory. The last 16 words are all one bits because the machine presently has only 24 K of memory.

To provide for faster processing a second table is used to keep track of the contiguous bits in the bit map. Each word corresponds to a word in the bit map. Each word has four fields:

LØ - Left justified zeros

RJØ - Right justified zeros

GØ - Greatest count of contiguous zeros

POS - Starting bit position of GØ

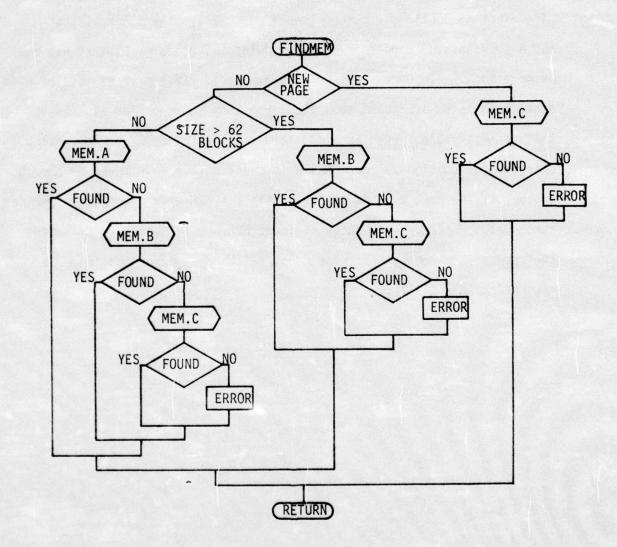


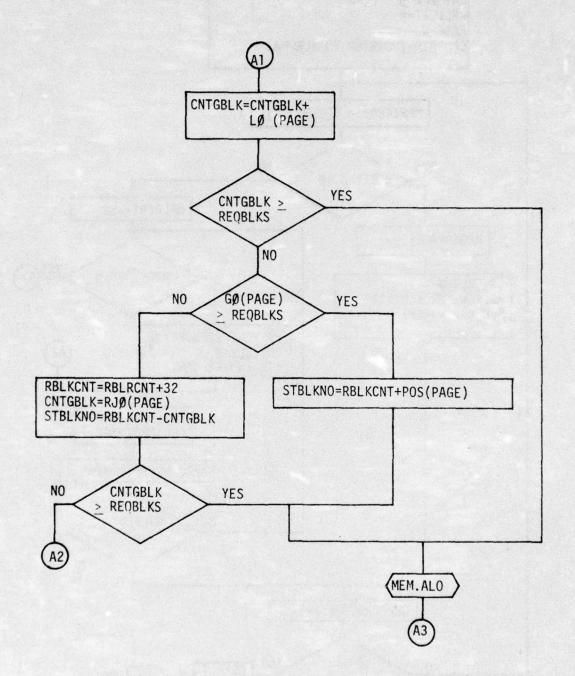
FIGURE 36 . FLOWCHART OF FINDMEM

#### MEM.A SERVICE SUBROUTINE

MEM.A searches memory accessible to a process to find the requested size block of contiguous memory (see figure 37). The outer loop of MEM.A searches for a pair of memory modules (16 pages each) that has one or more accessible pages (not write protected). If a pair of modules with accessible pages are found then the inner loop searches for each page that is accessible. When an accessible page is found the corresponding entry in the contiguous zeros table is checked to see if there is enough space. If the next page is also accessible MEM.A will attempt to combine space between them. When space is found MEM.ALO is called to allocate the memory. If memory space could not be found an error indicator is returned to the caller.

FIGURE 37 . FLOWCHART OF MEM.A

AD-A032 121 AUBURN UNIV ALA ENGINEERING EXPERIMENT STATION REAL-TIME PHASED ARRAY RADAR STUDIES. (U) 1976 E R GRAF, H T NAGLE F/G 17/9 DAAH01-71-C-1303 NL UNCLASSIFIED 2°F5 AB32121 뱂



#### MEM.B SERVICE SUBROUTINE

MEM.B searches for a free block of contiguous memory starting in accessible memory for a process (See figure 38). MEM.B examines each protect bit. When it finds one that permits access to a page it adds the number of left justified free blocks to a block count. If the block count is big enough for the request the memory is allocated by MEM.ALO and control is returned to the caller. If there is not enough memory and the page is not completely free the block count is set equal to the number of right justified free blocks and the starting position of the free blocks is updated. If the page is protected MEM.B checks to see if any unprotected free memory has been found and if the current page is completely free. If it is then MEM.B checks to see if enogh memory has been found. If not then MEM.B repositions the start of free blocks pointer and clears the block counter. If enough memory cannot be found MEM.B returns with an error code.

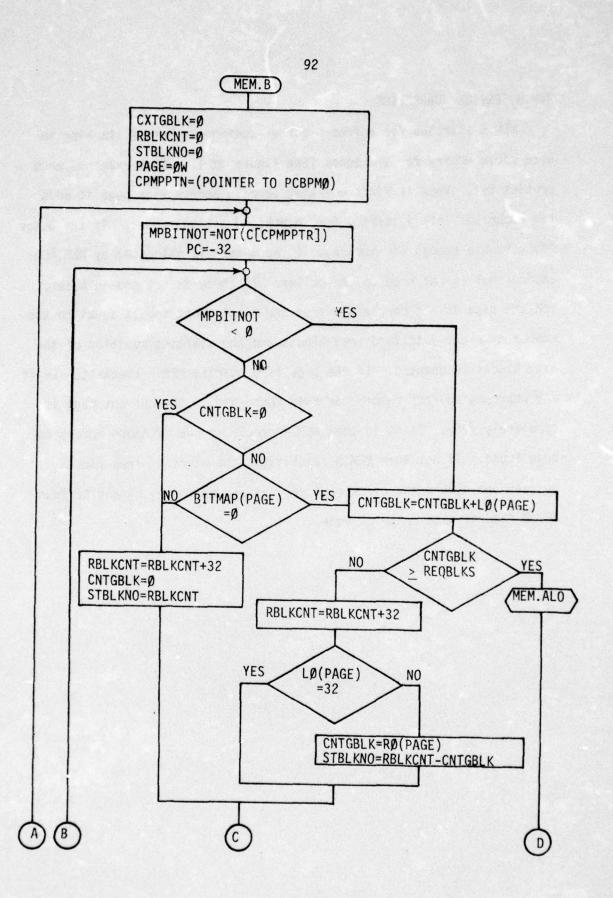


FIGURE 38 . FLOWCHART OF MEM.B

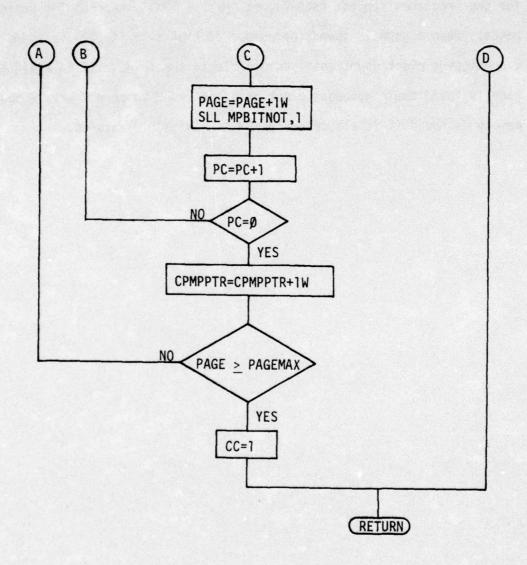


Figure 38 Continued

### MEM.C SERVICE SUBROUTINE

MEM.C searches unassigned pages for enough contiguous memory for the processes request (see Figure 39 ). MEM.C searches for unassigned pages. When a page is found the number of blocks in it (32) is added to a contiguous block count until enough blocks are found. If an assigned page is found the contiguous block count is reset to zero. When enough memory is found it is allocated by MEM.ALO and MEM.C returns.

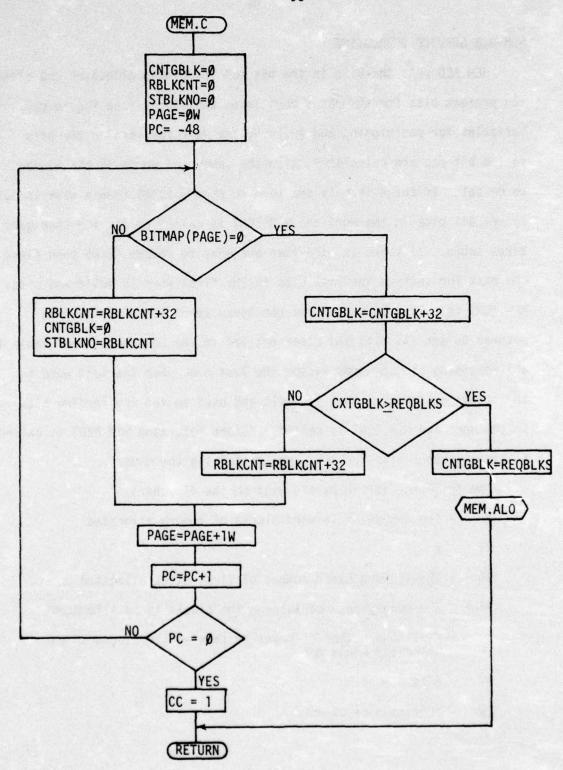


Figure 39 FLOWCHART OF MEM.C

#### MEM.ALO SERVICE SUBROUTINE

MEM.ALO sets the bits in the bit map for blocks allocated and clears the protect bits for the pages containing the blocks (see figure 40). Variables for positioning and builbing the mask for setting the bits in the bit map are calculated, also the number of words in the bitmap to be set. If there is only one word to change (J=Ø) then a mask is built to set all bits in the word and MEM.ZCNT is called to fix the contigous zeros table. If there is more than one word to change (J<Ø) then first the mask for setting the last bits in the first word is built and used. MEM.ZCNT is called to set up the contigous zeros table. A loop is entered to set all bits and clear entries in the contigous zeros table in all remaining bitmap words except the last one. For the last word in the bitmap to change a mask is built and used to set the leading bits in the word and MEM.ZCNT is called. Before returning MEM.PROT is called to clear protect bits for the pages containing the memory.

The following variables are used on the flowchart:

K - The number of 16 word blocks of memory allocated

K1 - K - 1

BD - The 16 word block number of first block allocated

PAGE - The memory page containing the blocks to be allocated

 J - Minus the number of pages containing the blocks to be allocated minus one

BW - bitmap word

ZW - contigous zeros word

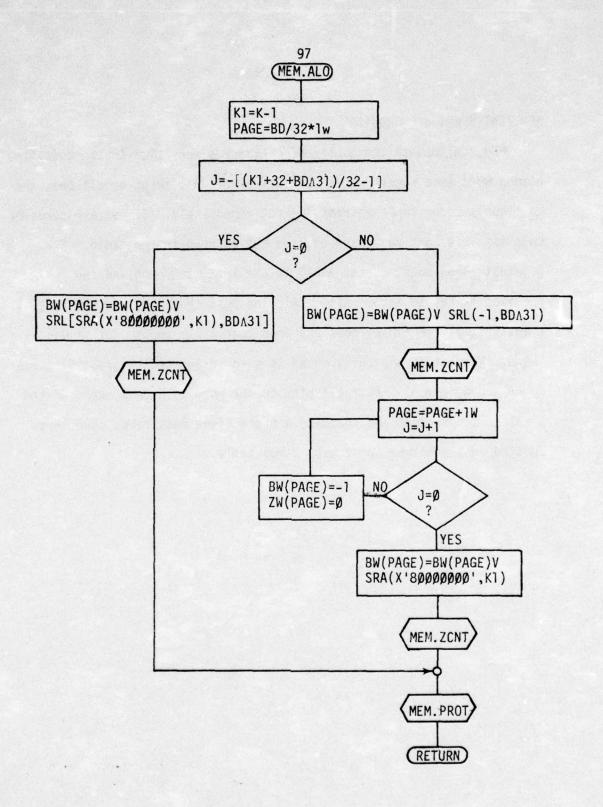
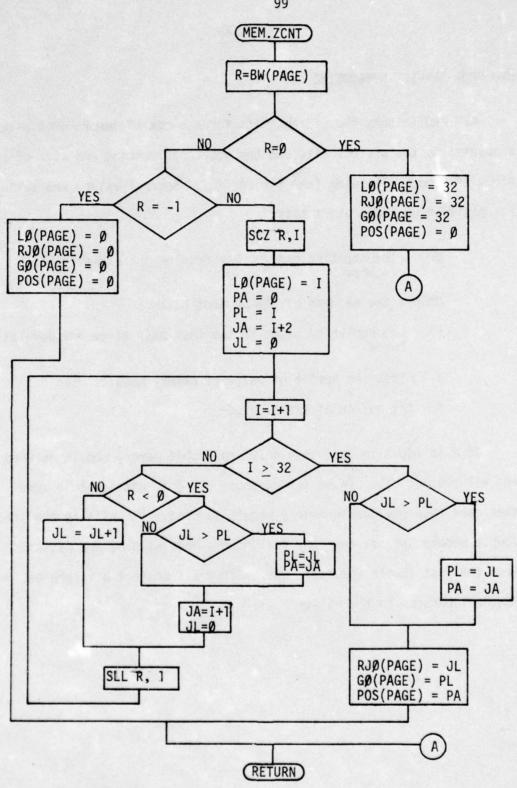


FIGURE 40 . FLOWCHART OF MEM.ALO

#### MEM.ZCNT SERVICE SUBROUTINE

MEM.ZCNT counts the number of contiguous zero bits in the specified bitmap word (see Figure 41). If the word is all zeros or all ones the contiguous zeros table entries are set immediately. If not all zeros or ones the left most zeros are counted and entered in the table. The greatest zeros count is set equal to the leftmost count and the position is set to zero. After skipping past the first one MEM.ZCNT scans past all remaining ones and then counts all zeros until a one is found. When it is the zeros count is used to update the greatest zeros count and position. After all bits in the word have been examined the greatest zeros count and position and the right most zeros count are updated in the bitmap contiguous zeros table.



I = BLOCK NUMBER (BIT POSITION IN BITMAP WORD)

JL = Contiguous zeros rightmost

R = Bitmapword

PA = Greatest number of zeros

PL = Position

Figure 41 FLOWCHART OF MEM. ZCNT

#### MEM. PROT SERVICE SUBROUTINE

MEM.PROT clears the protect bits for a block of memory when given a pointer to the protect bits and the starting location and size of the block of memory allocated (see figure 42). Several values are calculated for positioning the protect bits:

- BDD the starting page number from BD the starting block number
- DMOD the address of the protect bits
- KK the number of pages in the last pair of memory modules used
- J minus the number of pairs of memory modules used
- K the number of blocks used.

If J is equal to zero then only one double memory module is used and all protect bits can be set at once. If J is not equal to zero then more than one double memory module is used. The bits in the first double module are set then any full double modules used are set and last the last double module is set. After all protect bits are set up MEM.PROT returns to the caller.

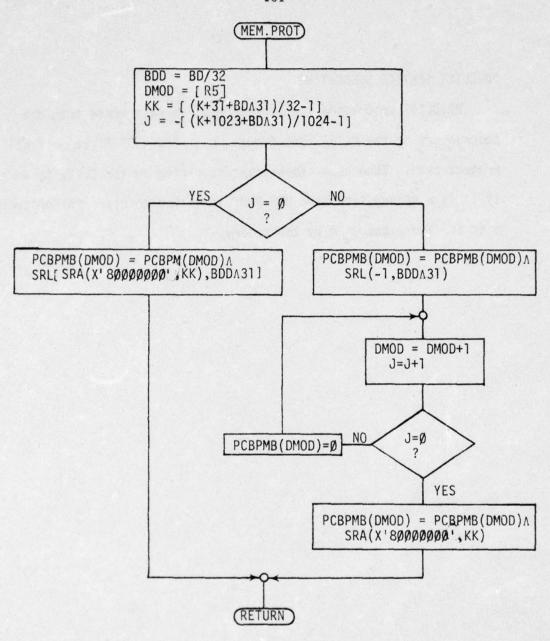


Figure 42 FLOWCHART OF MEM. PROT

## PB.CLIST SERVICE SUBROUTINE

PB.CLIST calculates the protect bits for a process from the descriptors on the CLIST (see figure 43). PB.CLIST first sets all protect bits. Then each capability descriptor on the CLIST is examined. If it is a memory type then MEM.PROT is called to clear the protect bits for the pages used by the memory.

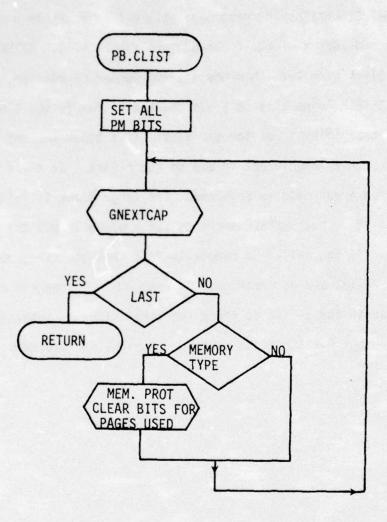


FIGURE 43 FLOWCHART OF PB.CLIST

# GET EMPTY CAPABILITY ENTRY ROUTINE (GETECE)

GETECE obtains the next available c-list entry in a chain of c-list blocks and adds a block if needed (see figure 44). GETECE first searches for a block with less than the maximum number of entries. If a block is found GETECE loops till it finds the empty slot in the block and returns with a capability index for the slot. If a block was not found then GETECE gets a large block to add to the c-list. If there are no large blocks an error code is returned. The large block is initialized to empty slots. Next GETECE searches for a place to put the block in the c-list. If the c-list is completly full then the large block is returned to the c-list and an error code is returned. If the end of the c-list or a gap in the c-list is found the large block is inserted there. An index number for the first slot in the block is returned.

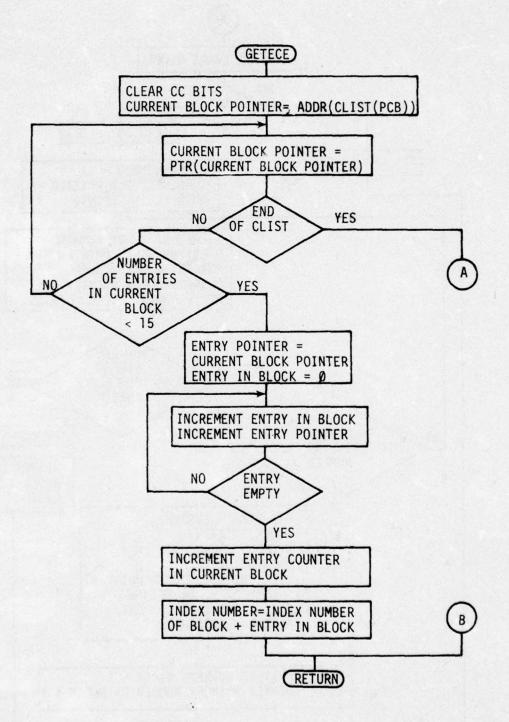


FIGURE 44. FLOWCHART OF GETECE

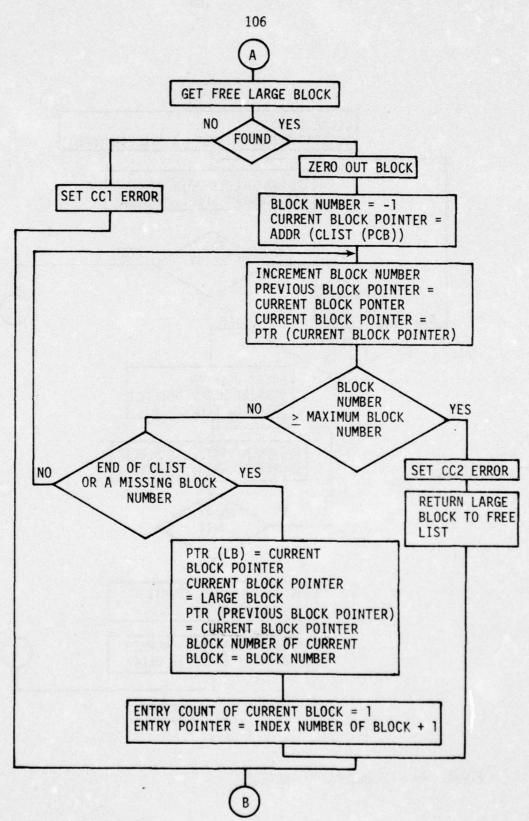


Figure 44 continued

## REMOVE CAPABILITY SERVICE SUBROUTINE (RMVCAP)

RMVCAP removes a capability from a c-list (see figure 45 ). RMVCAP gets the pointer in the c-list entry and returns to the caller if it is empty. The entry is cleared in the c-list. The entry counter counting the number of entries in the c-list block is decremented by one. If there are no entries left in the c-list block then RMVCAP removes the c-list block from the c-list and puts it on the free large block list. RMVCAP extracts the capability type from the pointer. If it is a son type RV.SON is called to remove the son from the system. If the type is a terminated son then RMVCAP returns to the caller because the son has already been removed. The user count in the capability descriptor is decremented by one. If there are any users left RMVCAP returns to the caller. If the capability type is greater than MAXTYP then the system is aborted through SYSABORT because the presence of an invalid capability type indicates that system data is corrupted. If the type is valid RMVCAP switches to a routine to remove that type of capability from the system. Last control is returned to the caller.

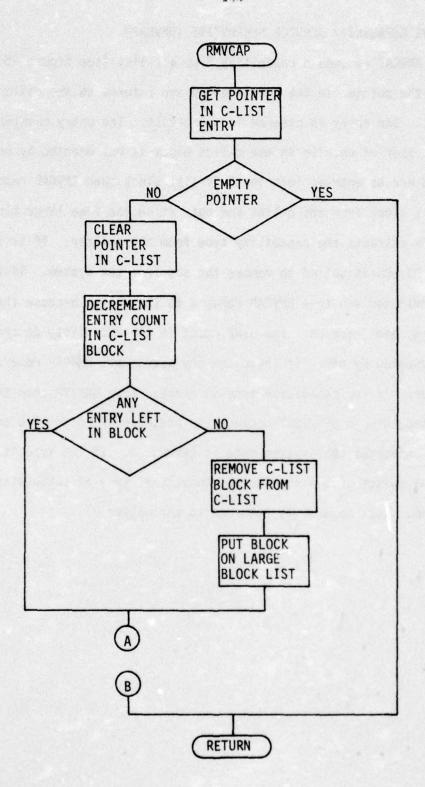


FIGURE 45 FLOWCHART OF RMVCAP

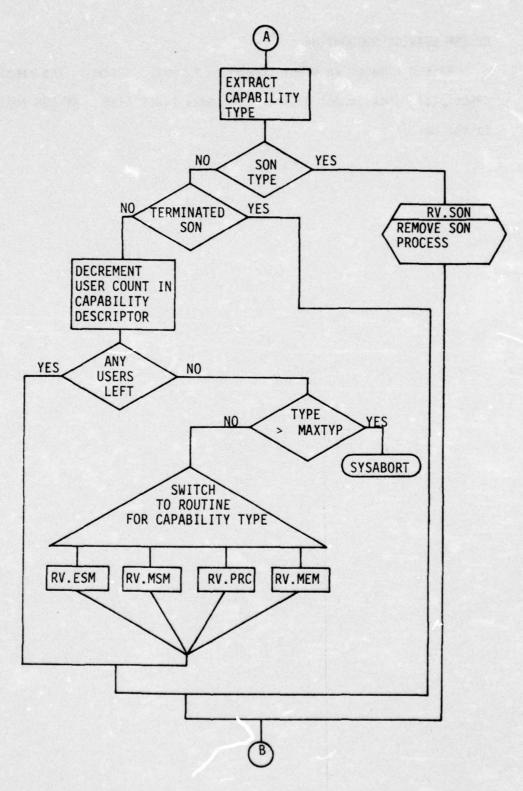


FIGURE 45. CONTINUED

# RV.ESM SERVICE SUBROUTINE

RV.ESM removes an event semaphore from the system. The semaphore descriptor block is put on the free small block list. RV.ESM returns to the caller.

### RV.MSM SERVICE SUBROUTINE

RV.MSM removes a message semaphore from the system after freeing any message blocks (See figure 46). DEQUEUE is called to get a message block from the list of messages on the semaphore queue. If there is a message block left it is put on the free small block list and RV.MSM loops back to dequeue the next block. When there are no message blocks left the descriptor block for the semaphore is put on the free small block list. Last RV.MSM returns to the caller.

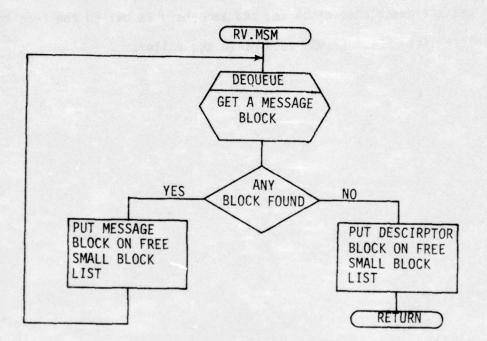


FIGURE 46. FLOWCHART OF RV.MSM

## RV.MEM SERVICE SUBROUTINE

RV.MEM removes a momory capability for a process (see figure 47).

FREEMEM is called to free the memory. PB.CLIST is called to go down the list of remaining capabilities of the process and calculate new memory protect bits. Last the descriptor block describing the capability is put on the free small block list.

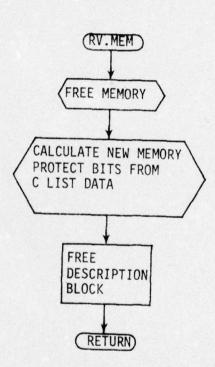


FIGURE 47. FLOWCHART OF RV.MEM

#### RV.SON SERVICE SUBROUTINE

RV.SON removes a son capability, terminates the son and flags the sons descendents for termination (See figure 48). FLGDSCND is called to put the descendent of detached process code in the PCB of the son and its descendents. The detached by father code is put in the sons PCB. RMVFMQ is called to remove the son process from the queue it is on. The code indicating what queue the son process is on in the PCB is changed to indicate the terminate queue and ENQUEUE is called to put it on the queue. Last VOP is called to signal the terminate process that a process is on the terminate queue.

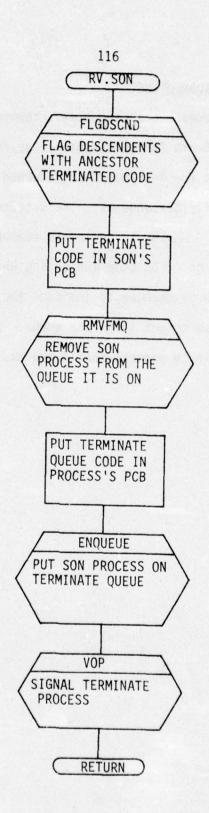


FIGURE 48. FLOWCHART OF RV.SON

# RV.PRC SERVICE SUBROUTINE

RV.PRC removes a procedure capability (see figure 49). FREEMEM is called to free the memory the procedure is in. Last the large block describing the procedure is put on the free large block list.

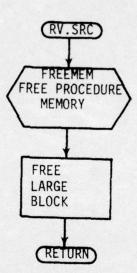


FIGURE 49. FLOWCHART OF RV.SRC

## FREEMEN SERVICE SUBROUTINE

FREEMEN releases memory by clearing the bits in the memory bit map and calculating a new contigous zero word (see Figure 50). From the starting address and length of the memory block to be freed the number of words in the bitmap is determined (|J| + 1). If there is only one word to change (J=0) then a bit string is built, using the bit position and length of the memory block, to AND with the word in the bitmap. Next a new contigous zero word is built by calling MEM.ZCNT. If there is more than one word to change in the bitmap then a mask is built to clear all bits in the first word starting at the first bit position of the memory block to be freed. Next the number of bits to clear is updated and the contigous zero word is recalculated by MEM.ZCNT. A loop is entered to update the bitmap words where the whole word must be cleared and the contigous zero word is set to a constant showing the maximum values. The loop is finished when the last word with bits to be cleared is left. The first N bits of the bitmap word are cleared, where N is the number of bits remaining to be cleared. Last the contigous zero word is updated by MEM.ZCNT.

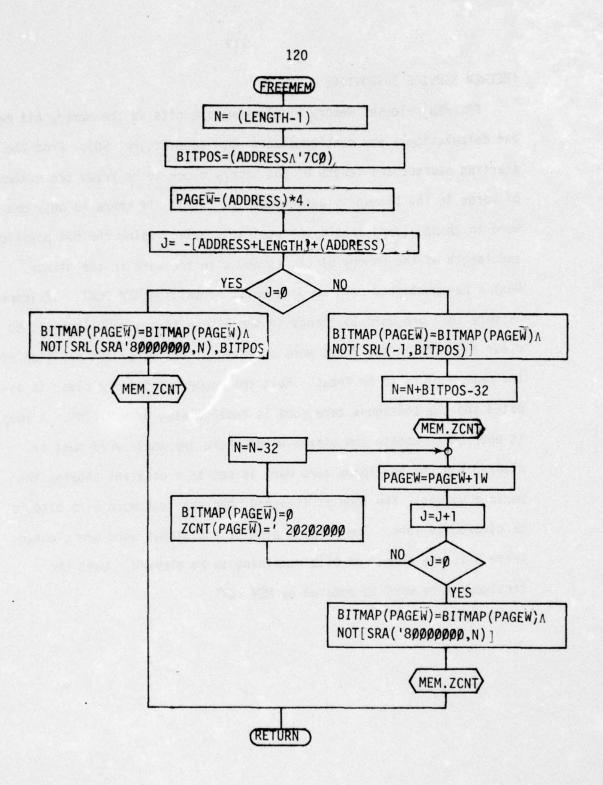


FIGURE 50. FLOWCHART OF FREEMEM

## FLAG DESCENDENTS ROUTINE (FLGDSCND)

FLGDSCND puts a termination code in the PCB of all descendents of a given process, including the given process (See figure 51).

FLGDSCND enters a loop to find all descendents of the process. GETDSCND is called to get the next descendent. The last descendent returned is the starting process. If the process returned is not the starting process the termination code is put in its PCB and the loop continues. If it is the starting process the termination code is put in and FLGDSCND returns to the caller.

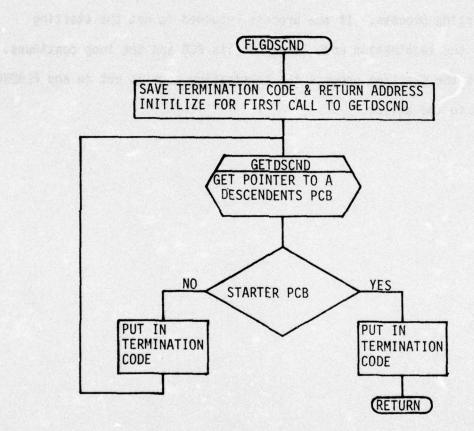


FIGURE 51 FLOWCHART OF FLGDSCND

## GET DESCENDENTS ROUTINE (GETDSCND)

GETDSCND finds all descendents of a process and makes them available to the calling program one at a time (see figure 52). The first time it is called GETDSCND sets up to start searching at the first capability of the process passed to it. If not the first time called GETDSCND backs up to the father of its current process and moves to the next capability entry of that process. A loop is entered to find the earliest descendent without any descendents. The loop is exited when the last capability entry in a c-list is found. If the current entry is not a son capability GETECE advances to the next entry. If it is a son capability the current entry pointer is saved in the PCB of GETDSCND's current process, the son process is made GETDSCND's current process, and the capability pointer starts out at the first entry for the current process. When the last entry in a c-list is found GETDCSND checks to see if its current process is the starting process. If it is then a condition code is returned. If not then a pointer to the current processes father is returned for the next call on GETDSCND.

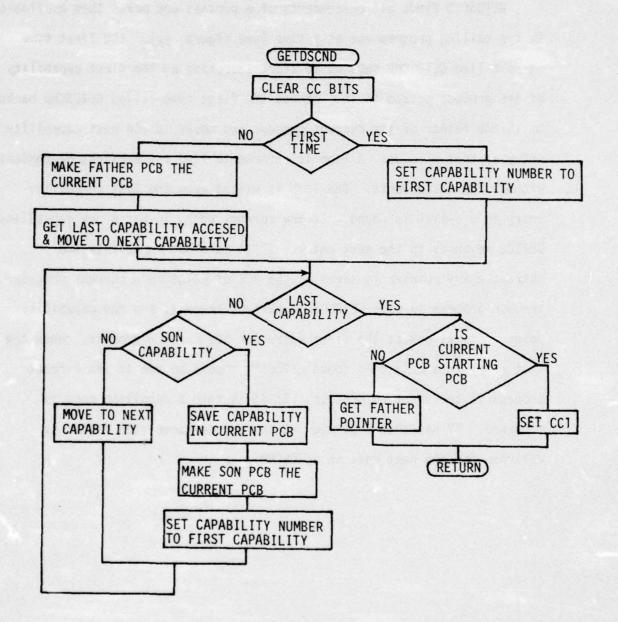
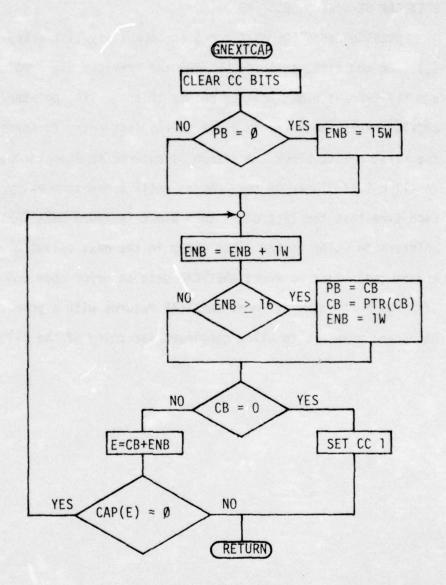


FIGURE 52. FLOWCHART OF GETDSCND

#### **GNEXTCAP SERVICE SUBROUTINE**

GNEXTCAP gets the next occupied capability list entry (see Figure 53). On the first call to GNEXTCAP the Previous Block pointer is zero and the Current Block pointer points to the c-list pointer in the FCB. GNEXTCAP sets the Entry iN Block to the last entry to force a move to the first c-list block. A search is entered to step through all entries in all c-list blocks of the process until a non zero entry is found. Each time that the last entry in a block is found GNEXTCAP moves the pointers to point to the first entry in the next c-list block. If the end of the c-list is found GNEXTCAP sets an error code and returns. If a non zero entry is found GNEXTCAP returns with a pointer to the entry and pointers to allow continued searching of the c-list.



Fiugre 53. Flowchart of GNEXTCAP

CLOSE FILE ROUTINE (IO.CLOSE)

IO.CLOSE closes a file for a process (see figure 54). IO.CLOSE will close a file for the process indicated by the caller but if blocking is required the current process will be blocked (required by the termination process). The File Control Block is valid if it points to a file descriptor which points back to it and the process. If the FCB is invalid IO.CLOSE returns with an error code. If the I/O busy flag shows no I/O in progress then the system capability for the memory containing the FCB is released and the file descriptor is cleaned up and put on the free file descriptor list. If the I/O busy flag was set IO.CLOSE checks to see if the I/O is finished. If it is then the system capability for the I/O buffer memory is released, and removed if the last user. Also the system capability for the FCB memory is released, and removed if the last user. If I/O is still in progress the blocked for close flag is set in the file descriptor so the SIGEND routine can finish the close operation. A condition code is set to indicate the current process is being blocked and POP is called to block the current process on the semaphore in the file descriptor.

Figure 54. Flowchart of IO.CLOSE

PROCESS SEND ROUTINE (PROSEND)

PROSEND sends a message to a semaphore on a given process's c-list (see Figure 55). FINDCAP is called to find the entry in the c-list.

If there is not a message semaphore capability found then PROSEND returns. If the semaphore counter is less than zero there are processes blocked on the queue. The message is put in the register save area of the PCB of the first process on the queue. VOP is called to unblock the process. If there are errors in VOP then SYSABORT is called because system data must be corrupted. If the semaphore counter was not less than zero then the message must be put on the semaphore queue. PROSEND gets a small block to put the message in. If there are no small blocks then PROSEND sets the cc bits to an error code and returns. If a block was found, PROSEND increments the semaphore counter. The message is put in the small block and is given the system message priority. ENQUEUE is called to put the block on the semaphore queue.

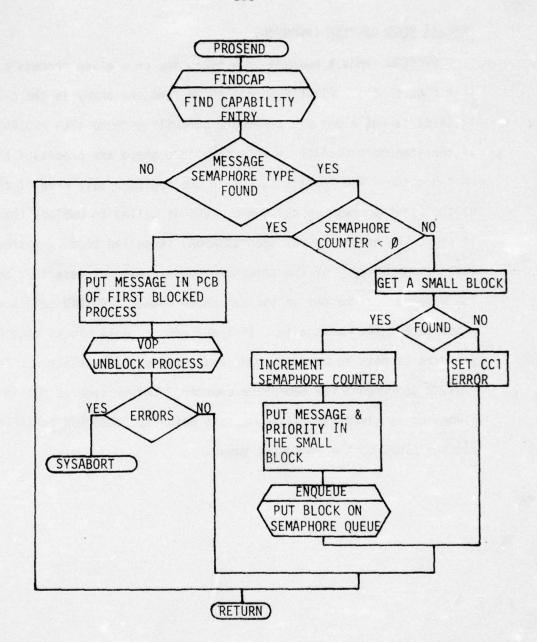


FIGURE 55. FLOWCHART OF PROSEND

I/O DEVICE SIGNAL END ROUTINE (SIGEND)

SIGEND acts as a special NUCLEUS call for I/O device handlers to signal I/O completion (see figure 56). SIGEND is called as a subroutine but it activates the CALM interrupt and sets up for a return through MONEXIT like a NUCLEUS call. The entrance to the NUCLEUS is necessary because SIGEND must work on system data and the file descriptor (see figure 18) for the I/O operation. If the process requesting I/O is not blocked on the semaphore in the file descriptor then SIGEND increments the semaphore counter and returns through MONEXIT. If the process is blocked a VOP is done on the semaphore. The system capability to the I/O buffer is removed. If the process was blocked for a close operation the close operation is completed by clearing flags in the file descriptor, removing the system capability to the memory containing the File Control Block, and freeing the file descriptor by putting it on a free list. Last SIGEND returns through MONEXIT.

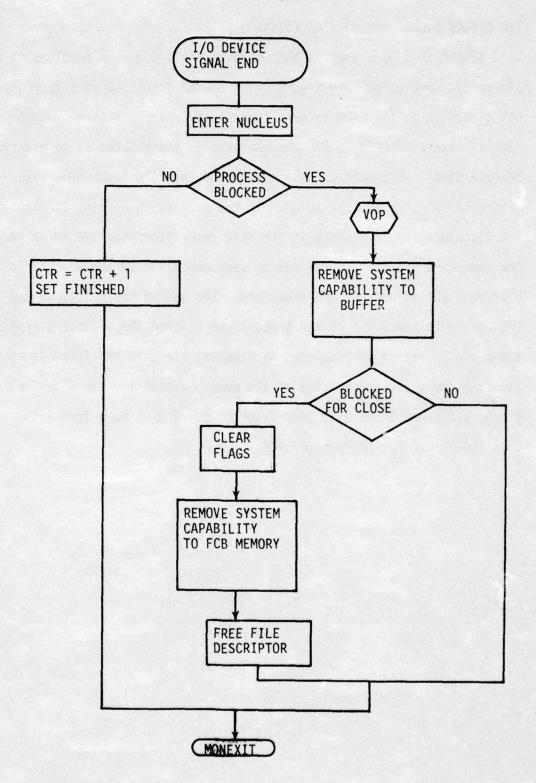


Figure 56. Flowchart of SIGEND

# GET INSTRUCTION ROUTINE (GETINS)

The get instruction routine (GETINS) is called by an interrupt routine to find the instruction that caused the interrupt. There are 9 possible cases where the instruction can be found, as diagrammed in Figure 57. The flowchart for GETINS is in Figure 58.

In case 1 the PSW points to the right half of a word so the last instruction had to be a halfword instruction in the left half. In case 2 the PSW points to the next word and because the right halfword is not a NOP the instruction would have been executed and a status bit in the PSW would be set to show that the last instruction was in the right halfword. In case 3 the PSW points to the next word and because the instruction in the right halfword is a NOP the left halfword instruction would have been executed as a word instruction and the status bit in the PSW would be reset, so the first bit of the instruction is examined to see if the instruction is a halfword instruction. In case 4 the PSW points to the next word, the status bit indicates a fullword was used, the instruction is a word instruction and is not an execute instruction. In cases 5 through 9 the PSW indicates a word instruction, which is an execute instruction. Information in the execute instruction is examined to find the instruction executed.

After the instruction actually executed is found it is extracted and the opcode is extracted from it. GETINS then returns to the caller with the opcode and instruction.

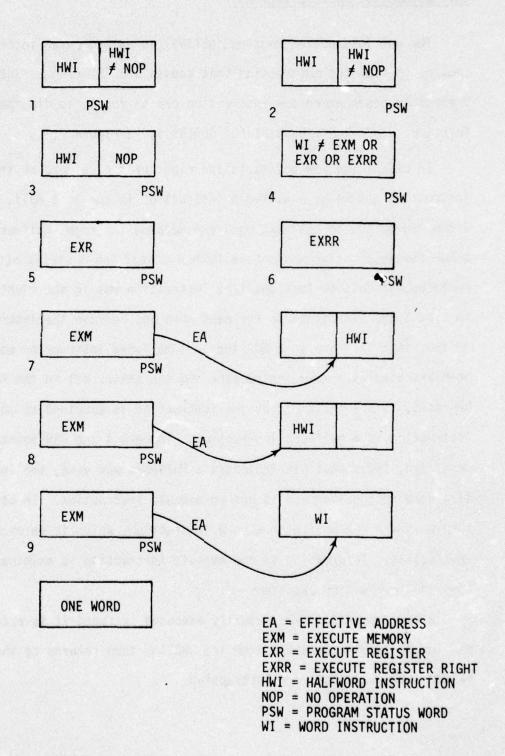


FIGURE 57. POSSIBLE CASES OF GETINS INPUT

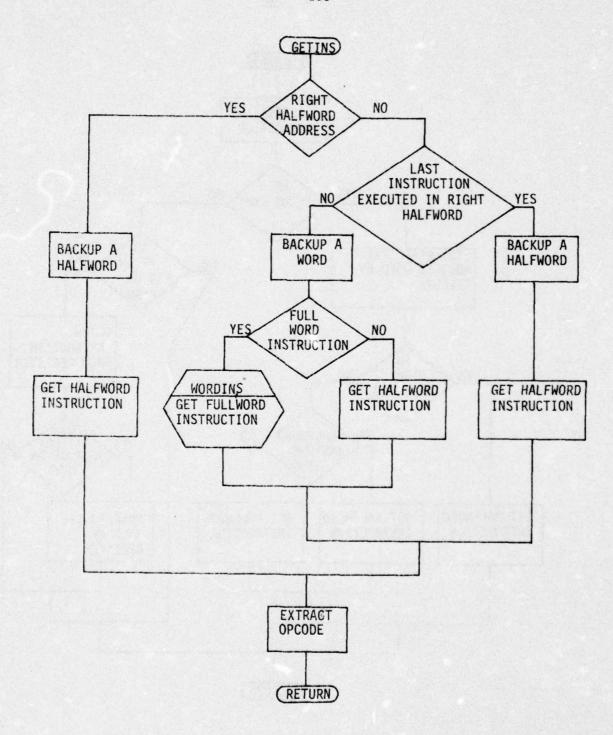


Figure 58. Flowchart of GETINS

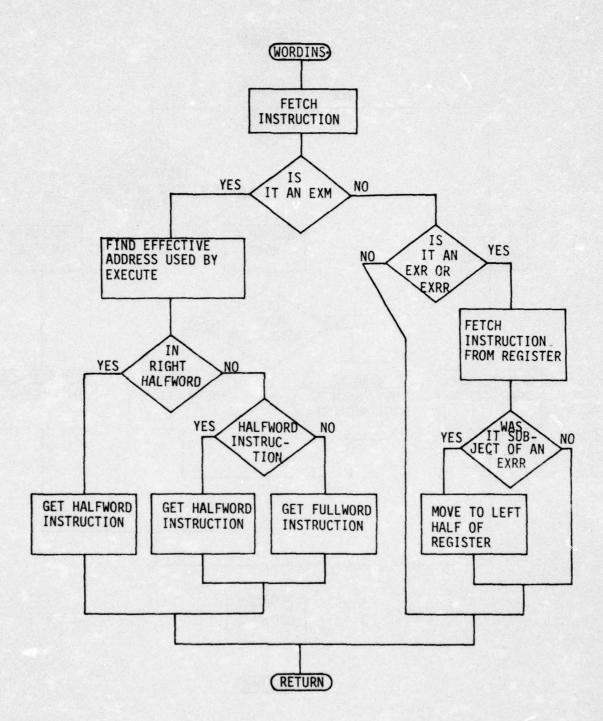


FIGURE 58. CONTINUED

## TERMINATION DELEATE ALL CAPABILITIES ROUTINE (TDACAPS)

TDACAPS is called by the termination process to remove all capabilities attached to a process (See figure 59). TDACAPS searches for all entries in the c-list. GNEXTCAP is called to find the next entry in the c-list. If none are left TDACAPS returns to the caller. If a capability is found TDACAPS enters the NUCLEUS by activitating the CALM interrupt level. RMVCAP is called to remove the capability. TDACAPS leaves the NUCLEUS by deactivating the CALM interrupt level. Next TDACAPS loops back for the next capability.

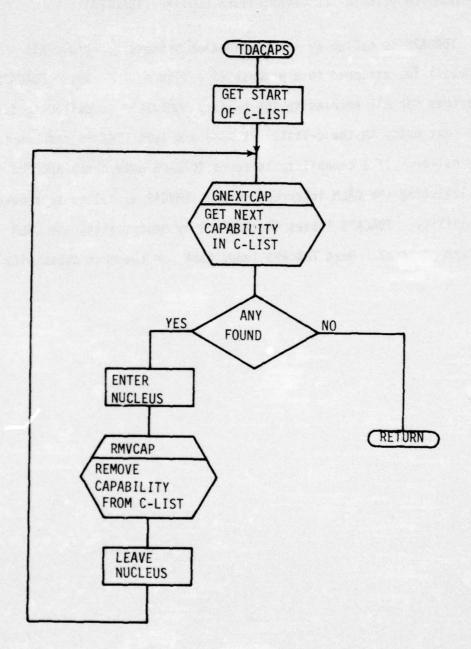


FIGURE 59. FLOWCHART OF TDACAPS

# TERMINATION FREE LARGE BLOCKS (TFLBLKS)

TFLBLKS is called by the termination process to free the large blocks used by the PCB (See figure 60). TFLBLKS enters the NUCLEUS by activating the CALM interrupt level so that it may work on the system data without being interrupted. If there are any return blocks they must be freed. TFLBLKS uses the pointer in the PCB to get the first return block. The return block is put on the free large block list and the pointer to the next return block is obtained. If there are any return blocks left TFLBLKS loops back to free the next one. After any return blocks are freed the second PCB block is freed. Next the first PCB block is freed. TFLBLKS leaves the NUCLEUS by deactivating the CALM interrupt level. Last TFLBLKS returns to the caller.



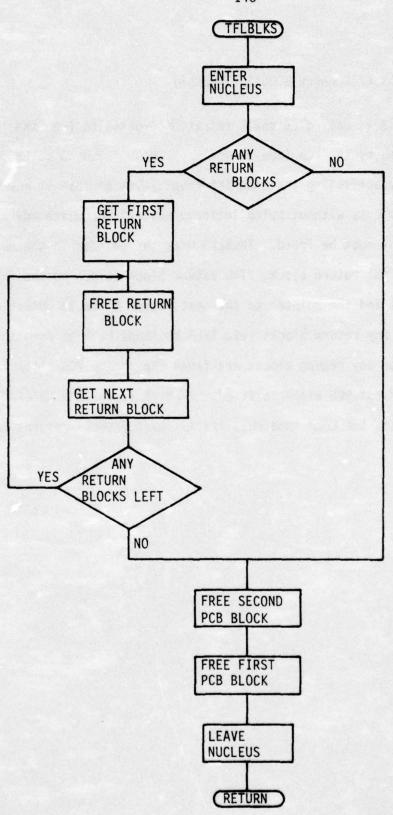


FIGURE 60. FLOWCHART OF TFLBLKS

## TERMINATE PROCESS CLOSE ROUTINE (TCLOSE)

TCLOSE is called by the termination process to close all open files for a process (see figure 61). A search is made of all file descriptors to find which ones are open for the process. If a descriptor is open to the process TCLOSE enters the NUCLEUS to close it. The CALM level interrupt is activated, registers are saved in the PCB and a return PSW is set up. IO.CLOSE is called to perform the close operation. The NECLEUS is exited by going through MONEXIT. If I/O was in progress IO.CLOSE will block the current process (TCLOSE) and select a new current process. When the close operation is finished control will be returned to TCLOSE. After any processing on a file descriptor TCLOSE moves to the next descriptor. If all file descriptors have been examined TCLOSE returns to the caller.

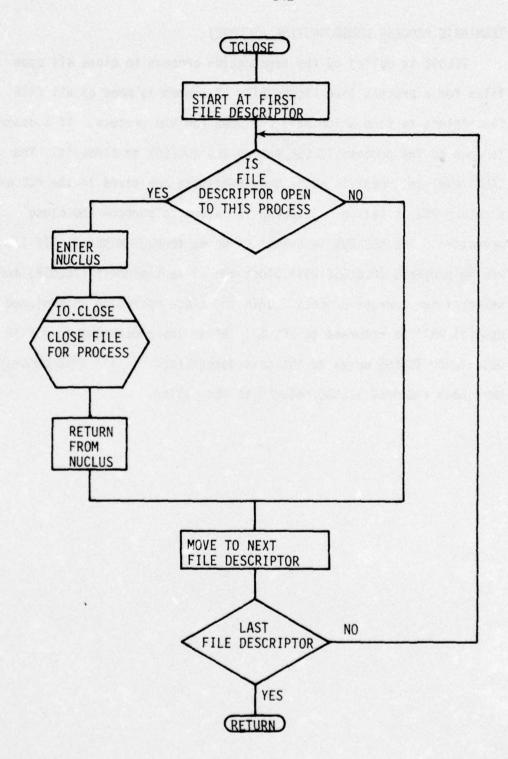


FIGURE 61. FLOWCHART OF TCLOSE

# CLEAR INTERRUPTS ROUTINE (CLRINT)

Several routines call CLRINT to clear out all interrupts when they abort the operating system (See figure 62). Starting with the lowest priority level (128) CLRINT disables and deactivates each interrupt level until it reaches the highest priority level (1). After all interrupts are cleared CLRINT returns to the caller

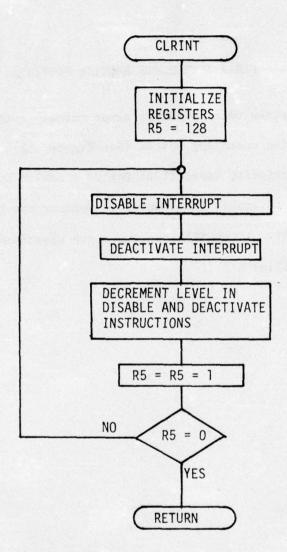


FIGURE 62. FLOWCHART OF CLRINT

### ABNORMAL TERMINATION SUBROUTINE (ABTERM)

The Abnormal Termination subroutine is called to abort the current process (see Figure 63). First the process is checked to see if it is essential, if it is then the system connot run without it, so SYSABORT is called to abort the system. Next the responsible ancestor for the process is found. The system starting process is a responsible ancestor so that if no responsible ancestor can be found for a process then the system data has been corrupted. When this condition is found SYSABORT is called to abort the system. After the responsible ancestor is found the responsible ancestor's C-LIST entry for the subtree of processes containing the aborting process is found. If the C-LIST entry cannot be found then system data is bad and SYSABORT is called to abort the system. Next the C-LIST entry is checked to see if it is a son capability. If not then SYSABORT is called because of bad system data. If all is well then the C-LIST entry is changed to a terminated son type. The process being aborted is put on the HOLD queue and the responsible ancestor's son is put on the terminate queue. A new current process is selected to take the place of the aborting process. If the responsible ancestor has a message queue for abort messages then a son abort message is sent to it. Next an abort code is put in the PCB's of all the processes on the aborting subtree and a different code is put in the process that caused the

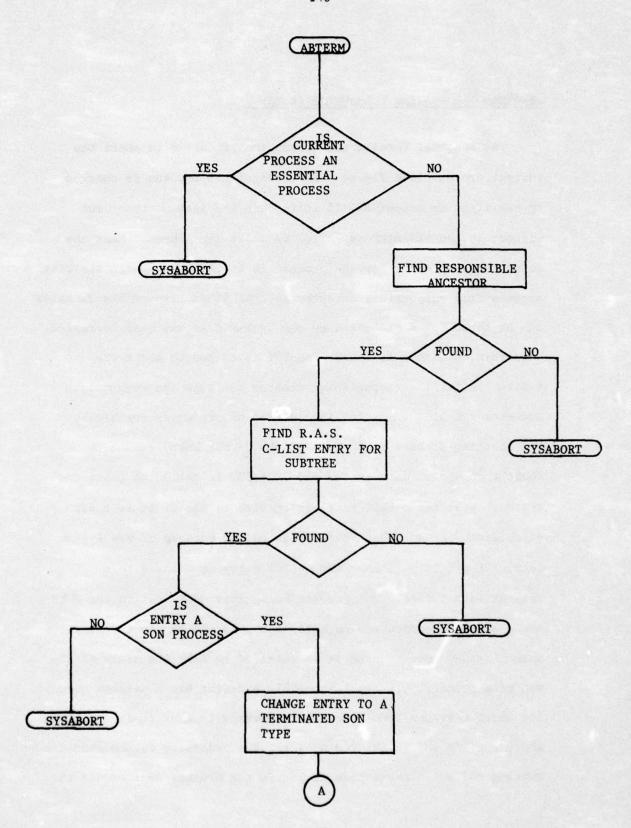


FIGURE 63. FLOWCHART OF ABTERM

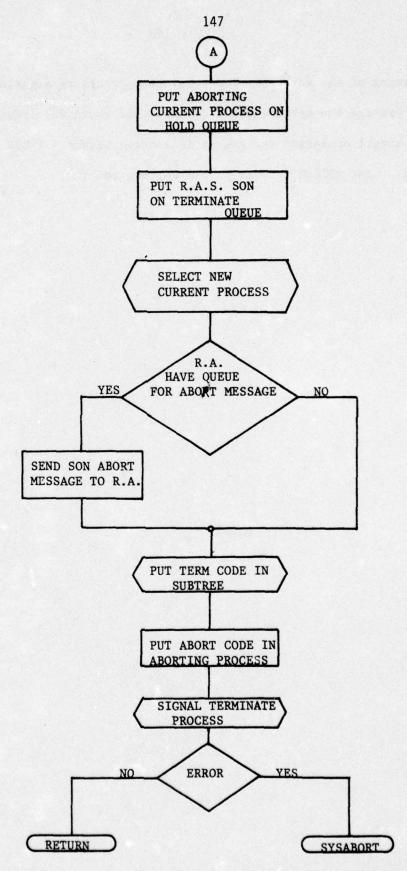


FIGURE 63. CONTINUED

aborting of the subtree. The terminate process is signaled so it can process the terminating processes. If there were any errors in the signal operation the system is aborted because of bad system data. Last ABTERM returns to the calling routine.

### SYSTEM ABORT ROUTINE (SYSABORT)

SYSABORT aborts the system when something goes wrong with the system (see Figure 64). CLRINT is called to disable and clear all interrupts. PRNREGT is called to print an error message, the contents of a register save area, and other data passed to SYSABORT by the aborting routine. SYSDUMP is called to dump system tables. Last SYSABORT goes to MASTER to restart the system.

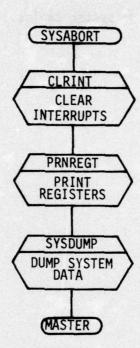


Figure 64. Flowchart of SYSABORT

#### PRINT REGISTERS ROUTINE (PRNREGT)

Several routines call PRNRECT to display the registers and PSW when they abort the operating system (See figure 65). Information passed in the registers tells PRNRECT where a header message is at and where the registers and PSW is saved at. The header message is typed on the teletype. Next the registers and PSW are converted to hex ASCII strings and printed. Last PRNRECT returns to the caller.

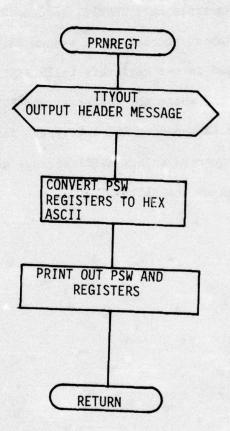


FIGURE 65. FLOWCHART OF PRNREGT

# SYSTEM DUMP ROUTINE (SYSDUMP)

SYSDUMP is called to dump the system data when the system is aborted. The present module is only a dummy and only prints an 'I GOT HERE' message.

VI. SYSTEM PROCEDURES

LOADER SERVICE PROCEDURE (LOADER)

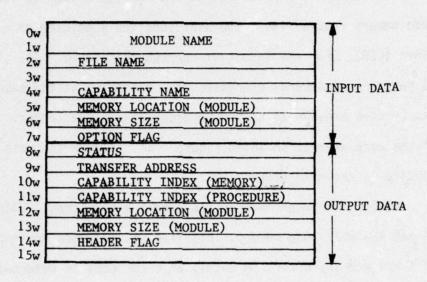
The loader is a procedure called through a CALM to load modules into memory from a file. The user supplies a pointer to a Loader Control Block (LCB). For the layout of the LCB see Figure 66. The loader is rentrent so several processes may be loading at the same time. A memory work area is allocated on each call to the loader. The layout of the work area is shown in Figure 67. LOADER the main loader routine is shown in Figure 68.

LOADER sequences through the loader subroutines. LDINIT is called to get the workspace memory, initilize it, and open the input file.

If there are any errors in LDINIT an error code is returned to the caller. FND.MOD is called to scan through the input file till it finds the requested module. If the module is not found an error code is returned to the caller. MEM.OPT is called to find memory to load the module in.

If not enough memory is found an error code is returned to the caller.

LOD.MOD is called to load the module into memory. If any errors occur an error code is returned to the caller. If the caller wanted a procedure the monitor is called to change the memory to a procedure. If any errors occur an error code is returned to the caller. Last the workspace is freed and control is returned to the caller.



#### OPTION FLAG:

- BIT #6 LOADED MODULE TO BECOME A PROCEDURE
- BIT #1 MEMORY HAS BEEN SUPPLIED BY USER
- BIT #2 IF USER SUPPLIED MEMORY INADEQUATE, FIND MEMORY
- BIT #3 MEMORY IS TO BE STARTED ON A NEW PAGE

#### STATUS:

- BIT #6 WORK SPACE MEMORY NOT FOUND, OPEN FILE ERROR
- BIT #1 MODULE NOT FOUND IN SPECIFIED FILE
- BIT #2 MODULE SPACE IN CORE NOT FOUND
- BIT #3 I/O ERROR INVALID TYPE CODE OR UNEXPECTED EOF
- BIT #4 ERROR IN MAKING LOADED MODULE INTO A PROCEDURE

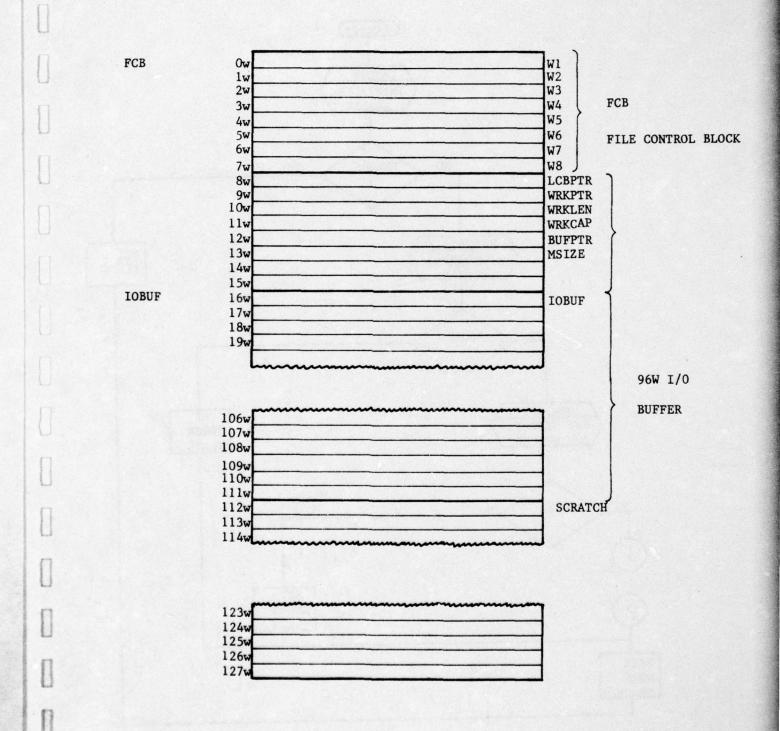


FIGURE 67. LOADER WORKSPACE

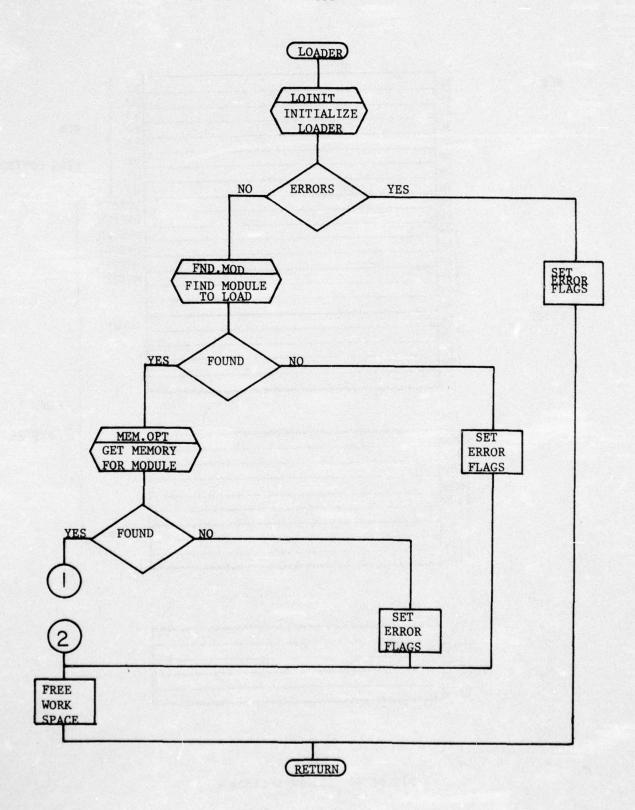


FIGURE 68. FLOWCHART OF LOADER

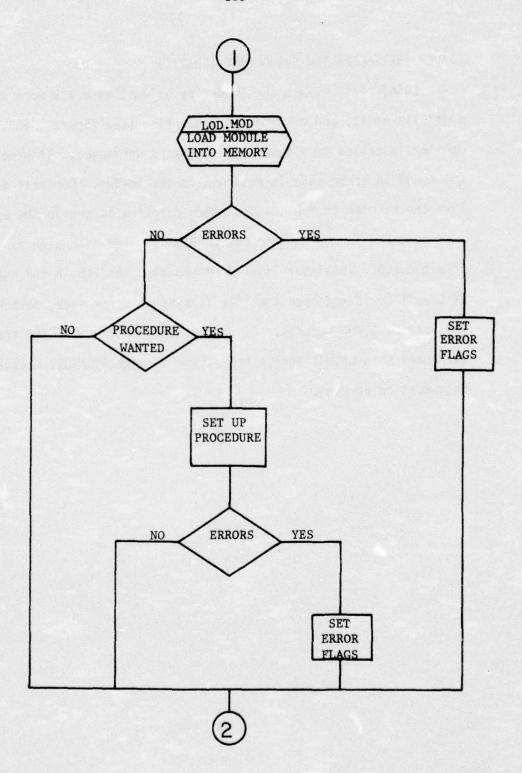


FIGURE 68. CONTINUED

### LOADER INITIALIZATION SUBROUTINE (LDINIT)

LDINIT initializes the loader by allocating a 128 word workspace, initializing it, and opening the input file (see figure 69). The monitor is called to allocate the 128 word workspace. If memory is not found an error code is returned to the loader. Constant data like the pointer to the LCB and other pointers is put in the workspace. A default FCB is copied into the workspace. The file name is copied form the LCB. The input file is opened with the FCB in the workspace. If LDINIT could not open the file it sets an error code, detaches the workspace and returns. If errors occur while detaching the workspace then LDINIT aborts the system through SYSABORT because system data must be corrupt.

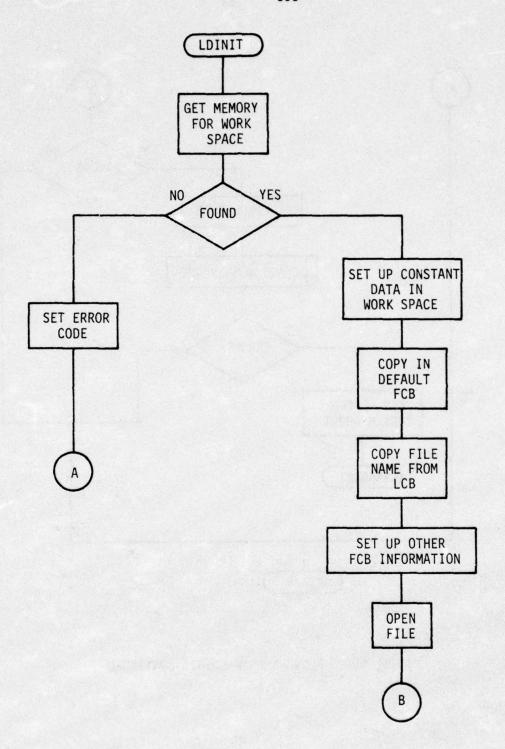


FIGURE 69. FLOWCHART OF LDINIT

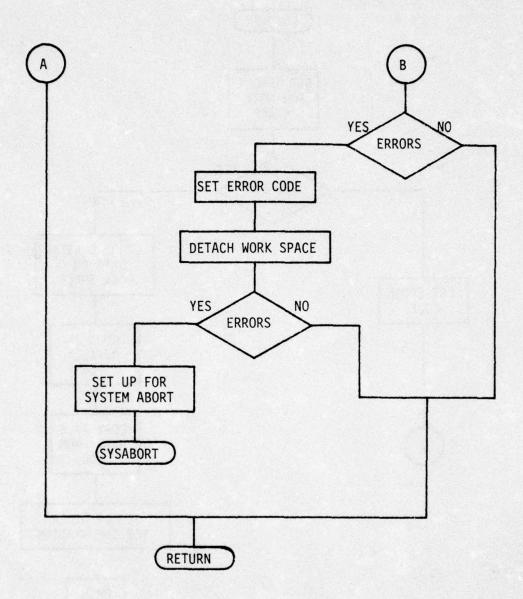


FIGURE 69. FLOWCHART OF LDINIT CONTINUED

#### LOADER FIND MODULE SUBROUTINE (FND.MOD)

FND.MOD reads records until the correct module name is found in the specified file (see figure 70). If a read error occurs an error code of one is returned to the loader. If an end of file is found then an error code of four is returned to indicate the module was not found. If the first item in the record is not a program name then FND.MOD continues to search for the start of the module. If the program name in the first item is not the desired name then FND.MOD starts searching for the start of the next module. When the module is found then the module size is obtained. If the proper type of items do not follow the program name then the size cannot be found and an error code of two is returned to the loader. If the module was found then the module size is returned in the workspace memory.

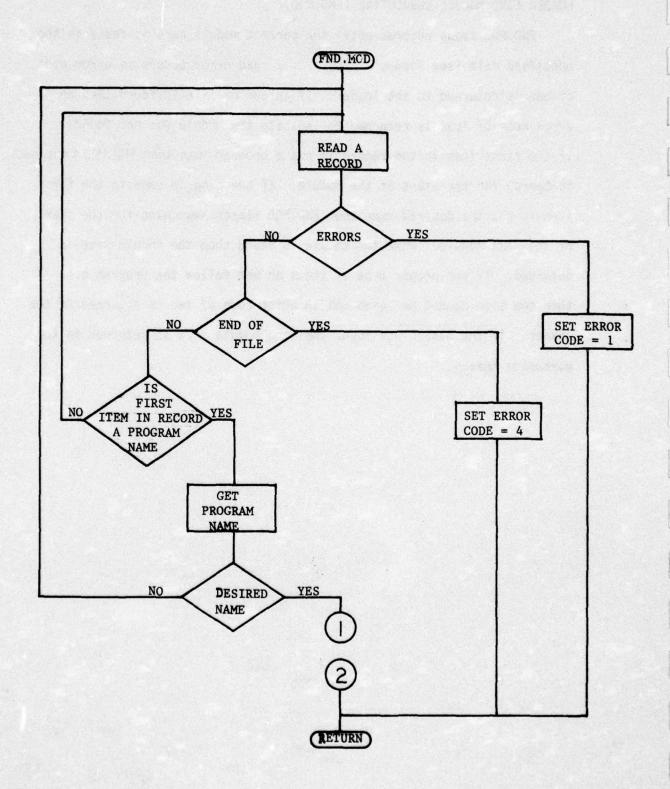


FIGURE 70. FLOWCHART OF FND. MOD

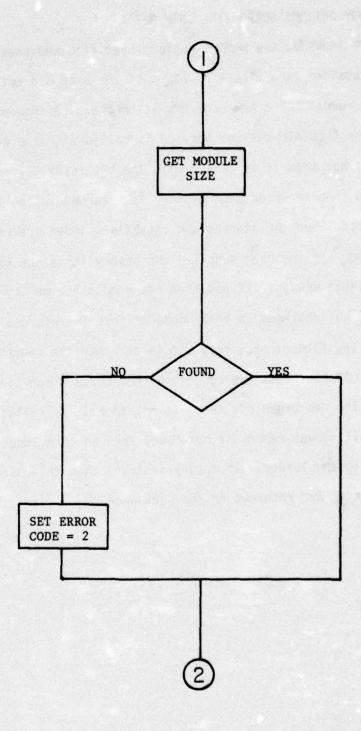


Figure 70. Continued

LOADER MEMORY OPTIONS SUBROUTINE (MEM.OPT)

MEM.OPT looks at the memory options specified and sees that enough memory is obtained (see figure 71). If the user did not supply memory the request for a new page bit is checked. If new pages was requested the ALLOCATE monitor service is called with the new pages bit set. If not then it is called with the bit reset. If enough memory is not found then an error code of four is returned to the loader. If memory is found then the address and capability index are returned in the workspace. If the user supplied the memory it is checked to see that it is large enough. If not then the find other memory bit is checked. If not set then an error code of four is returned to the loader. If the find other memory bit is set then the request for a new page bit is checked. If new pages was requested the ALLOCATE monitor service is called with the new pages bit set. If not then it is called with the bit reset. If enough memory is not found then an error code of four is returned to the loader. If memory is found then the address and capability index are returned in the workspace.

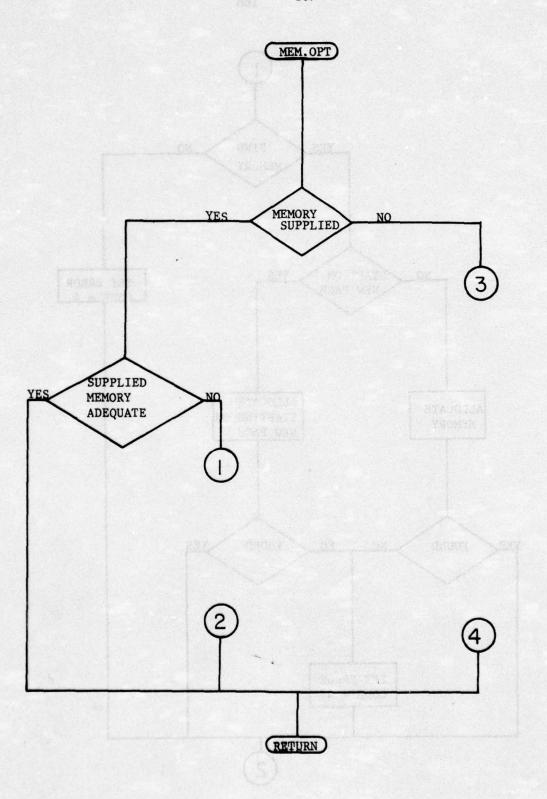


FIGURE 71. FLOWCHART OF MEM.OPT

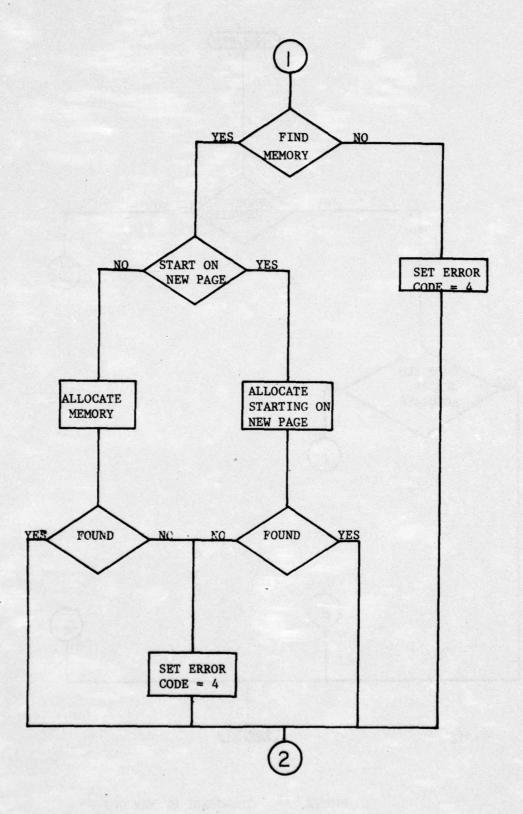


Figure 71. Continued

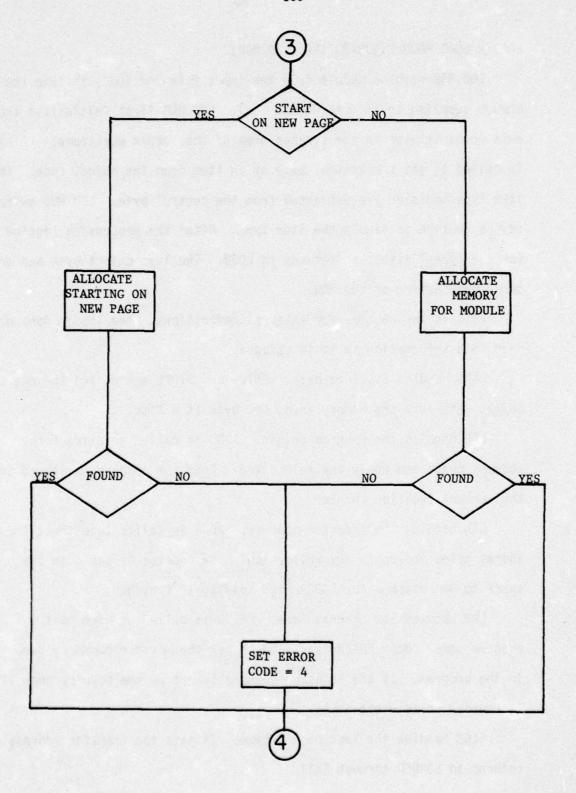


Figure 71. Continued

LOADER LOAD MODULE SUBROUTINE (LOD.MOD)

LOD.MOD reads a module from the input file and loads it into the memory supplied to it (see Figure ). LOD.MOD first initializes the data words it uses in the scratch area of the loader workspace. FETCH is called to get the control byte of an item from the object code. The item type and size are extracted from the control byte. LOD.MOD switches to the routine to handle the item type. After the processing routine for the type finishes it returns to LO3Ø. The last object byte and errors cause termination of LOD.MOD.

LEXD is the routine for external definitions. The loader does not need this information so it is skipped.

LABS handles absolute data. FETCH and STORE are called to move the object data into the memory area, one byte at a time.

LPO handles the program origin. ADDR is called to extract the address bytes and apply any relocation. Next the address is stored in the current location counter.

LTA handles the transfer address. ADDR is called to extract the address bytes and apply any relocation. The address is saved in the workspace to be returned to LOADER when loading is finished.

LPN handles the program name. FETCH is called to move past the program name. Next FETCH is called to get the maximum boundary used in the program. If the location counter is not on the boundry then it is rounded up to where it is.

LLOB handles the last object byte. It gets the transfer address and returns to LOADER through EXIT.

LREL handles relocatable data. It checks to see that the data is an integral number of words or an integral number of words plus three bytes. FETCH is called to get the first three bytes or a word. The relocation bias is added to the data. STORE is called to put the data in memory. If there is any data left LREL loops back to FETCH for the next word.

ADDR is a subroutine to fetch an address from the object data. FETCH is called to get the three byte address. If the address is relocatable the relocation bias is added to it. Last ADDR returns to the caller.

FETCH gets the bytes of an item from the I/O buffer and reads in a new record when needed. If it is the first time it has been called the first record is already in the I/O buffer and FETCH processes it as if it were just read. If all of the data in the buffer has been used then a new record is read by IN.MOD. If an error occurs an error code is returned to the loader through ERH2. After a read the pointers to the data in the buffer are reset to the start. The useable byte count in the record is used to calculate a pointer to the end of the data. The checksum for the data is calculated and compared to the checksum in the record. If there is a checksum error an error code is returned to the loader through ERH5. The sequence number in the record is compared to a counter in the workspace and if not equal then an error code is returned to the loader through ERH6. If the buffer has data in it the bytes are extracted one at a time until all the data is gotten. When the data is extracted FETCH returns to the caller.

STORE puts data into the memory. The address is checked to see that it is in the correct memory block. If not then an error code is returned to the main loader routine through ERH7. The data is positioned in a register so that it may be stored one byte at a time. The byte is stored in memory and the address pointer and bytes remaining count are updated. If not the last byte the next byte is positioned in the register. If the last byte then STORE returns to the caller.

ERH2 - ERH7 set up the error codes for an error exit.

EEXIT sets the return condition code for an error exit.

EXIT returns to the main loader routine.

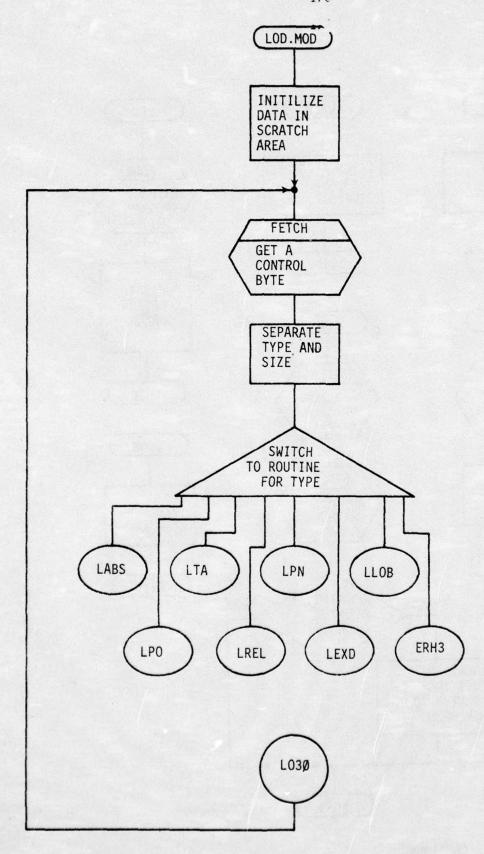


FIGURE 72. FLOWCHART OF LOD.MOD

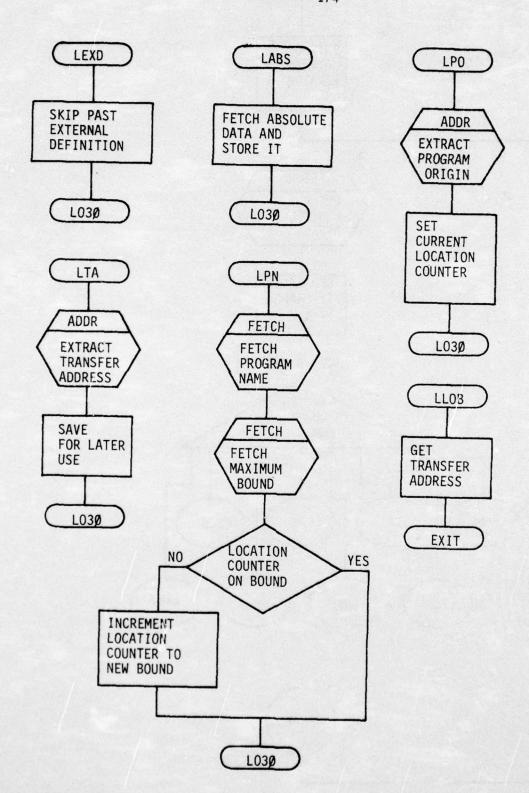
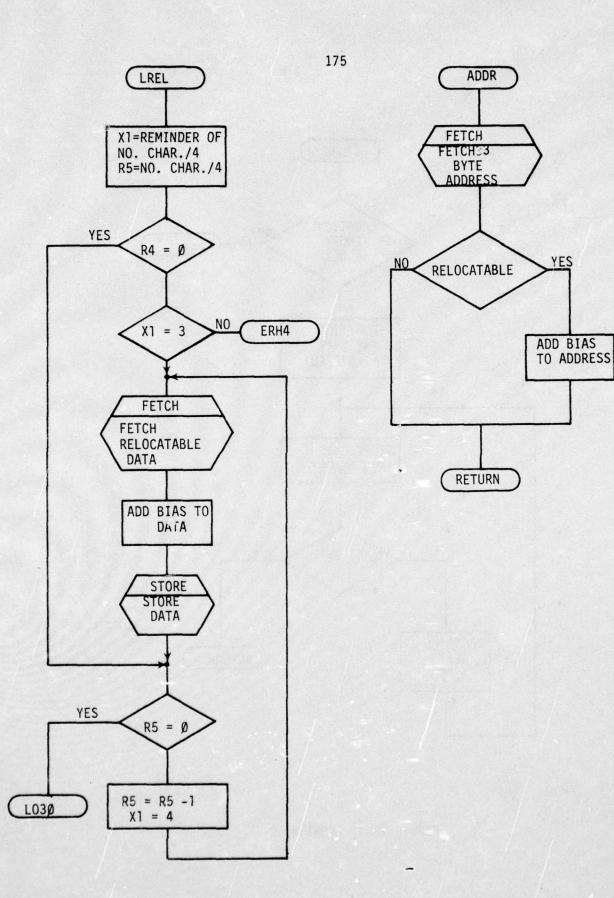


FIGURE 72. CONTINUED



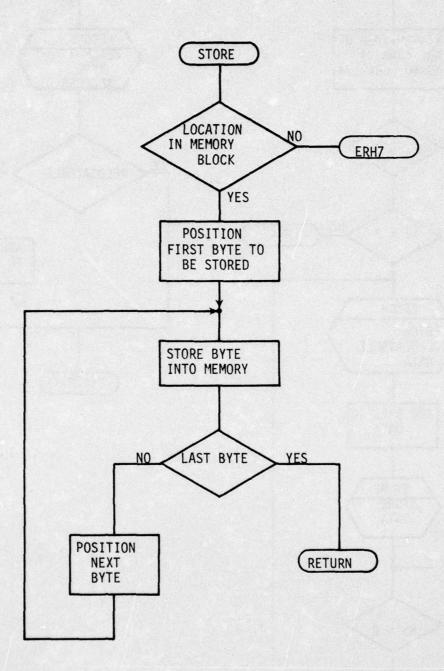


FIGURE 72. CONTINUED

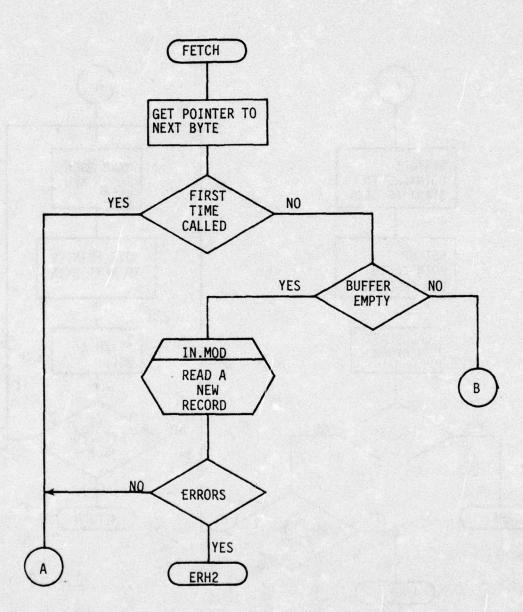


Figure 72. Continued

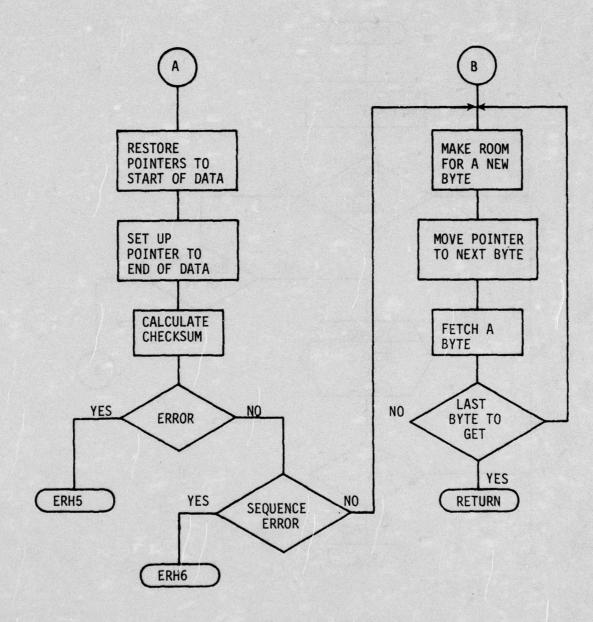


Figure 72. Continued

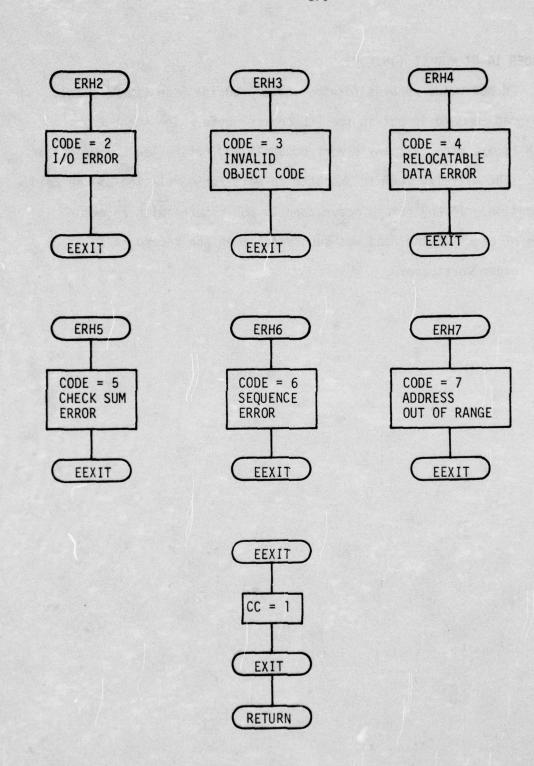


FIGURE 72. CONTINUED

#### LOADER INPUT MODULE (IN.MOD)

IN.MOD reads records for the loader routine (see figure 73). The read command is put in the FCB control byte. The start I/O CALM is executed. If any errors occur IN.MOD returns an error code of one. The wait I/O CALM is executed to wait for completion of the read operation. If any errors occur then IN.MOD returns with an error code of one. If the read was successful then the record is in the loader workspace.

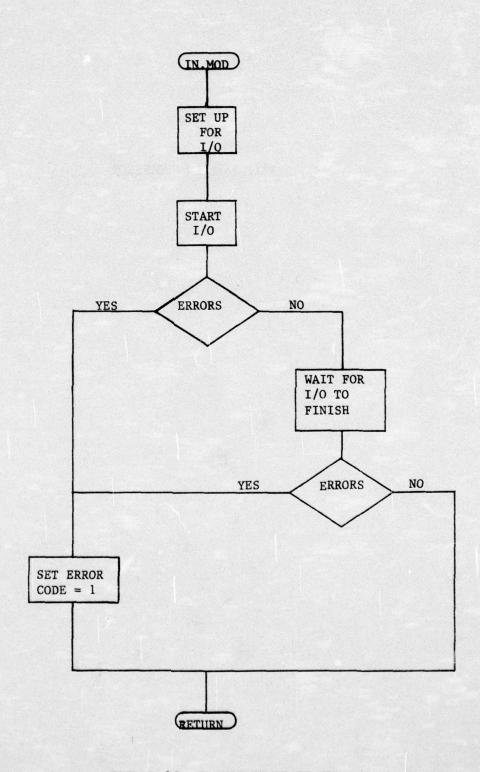


FIGURE 73. FLOWCHART OF IN. MOD

VII. SYSTEM PROCESSES

#### I/O PROCESS GENERAL

The device drivers for I/O devices are run as separate processes. They communicate with other processes through semaphores and system file descriptors. Each driver has its own semaphore from which it receives a message pointing to the file descriptor (see figure 18). At system initialization each I/O process should be set up and given the capabilities for its own semaphore and the system semaphores it makes use of.

The general operations for a device process is shown in figure 74. The first operation is to initialize the process and the device. Next it enters the loop it will execute till the system is shut down. At the the start of the loop the process waits on its message semaphore. It will receive a pointer to the file descriptor in register 4. In the file descriptor is a pointer to the users File Control Block, which contains the device command (see figure 83). The process next executes the command. If it must wait for an interrupt it should set up the interrupt location to point to the SI map down entry for the device (named SUBDCCxx where xx is the device number, the TTY is 7E). After the operation is complete SIGEND is called to signal the end of the operation on the semaphore associated with the file descriptor. Next the driver goes back to the start of the loop.

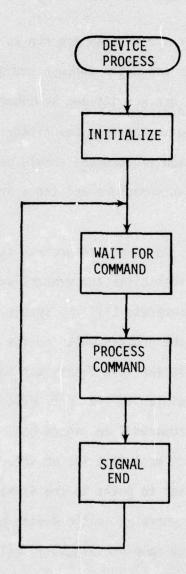


Figure 74. FLOWCHART OF A GENERAL DEVICE PROCESS

#### I/O PROCESS OPCINDV - OPERATOR COMMUNICATIONS INPUT

OPCINDV acts as a device handler for communicating to the operators. It can input messages from the operator communication devices and gives the message to the file indicated by the operator. If the file is not looking for any input when the message comes in one message will be queued up till it is asked for. The message from the operator starts with a four letter file name or a special character.

The flow chart for OPCINDV is in figure 75. OPCINDV must be set up as a process with appropriate capabilities by the system initialization routine. When OPCINDV starts it will do any necessary initialization.

Next it waits for a message from the operator input devices or an input command from a process. After a message or command comes it is checked to see which it is. If it is a command then OPCINDV switches to a routine to handle it. If it is a message then it is given to that file.

A command is first checked to see if it is valid. If not then status is set in the file control block and an end of operation is signaled to the file semaphore. If the command is a read command then the file descriptor is checked to see if a message is waiting. If a message is waiting then it is put into the buffer area specified by the file and the message buffer is freed. Next end is signaled to the file semaphore and OPCIND waits for the next message or command. If a message is not waiting then the waiting for message flag is set and OPCINDV waits for the next command or message. The other valid commands set appropriate status bits. Signal end to the file semaphore and start waiting for the next message or command.

AD-AU32 121 AUBURN UNIV ALA ENGINEERING EXPERIMENT STATION REAL-TIME PHASED ARRAY RADAR STUDIES. (U) 1976 E R GRAF, H T NAGLE F/G 17/9 DAAH01-71-C-1303 UNCLASSIFIED NL 3 OF 5 AB32121 

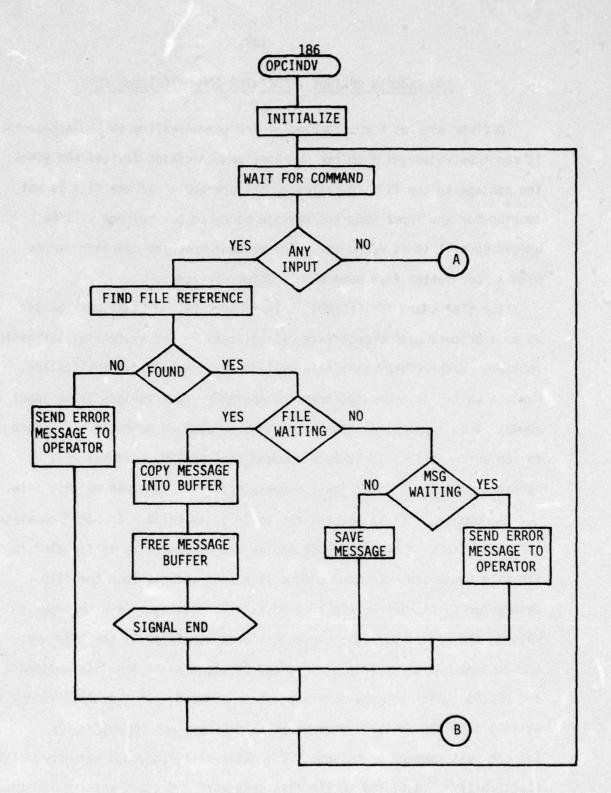


Figure 75. Flowchart of OPCINDV

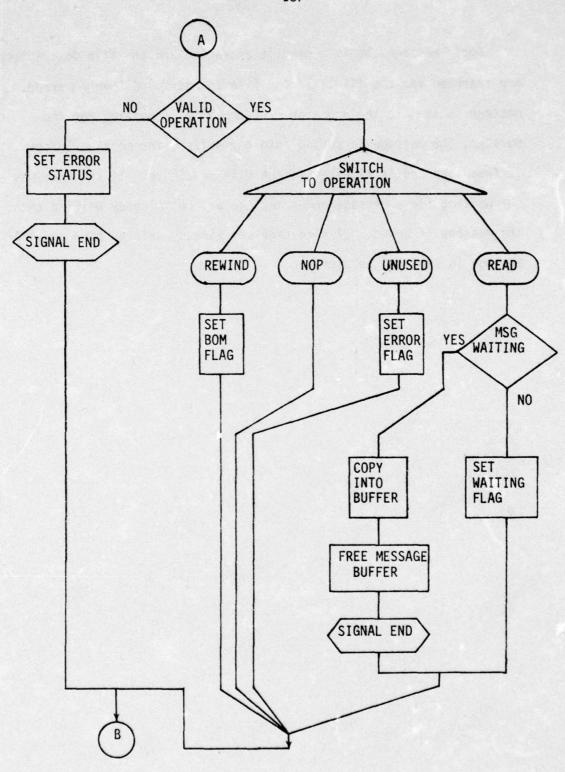


FIGURE 75. CONTINUED

For a message the file name is extracted and the file descriptors are searched for the file. If the file is not found then an error message is sent to the operator. If the file is waiting for the message, the message is copied into the buffer, the message buffer is freed and end is signaled to the file semaphore. If the file was not waiting for a message and a message was not already waiting then the message is saved. If a message was already waiting then an error message is sent to the operator.

## I/O PROCESS OPCOUTDY - OPERATOR COMMUNACTIONS OUTPUT

OPCOUTDV acts as a device for switching messages to the operator to the appropriate operator output device. The first byte in the message contains a code for which operator code Ø and any undefined codes go to the device currently defined as the primary operator device. Each device will be assigned its own code (teletype is 1).

The flow chart for OPCOUTDV is in figure 76. The OPCOUTDV process is set up at system initialization along with some of its capabilities. When it starts up OPCINDV first does some appropriate initialization. Next it enters a loop of looking for commands and executing them. The first thing in the loop is to wait on the OPCOUTDV command semaphore for a command. When the wait is finished the command opcode is checked and if invalid then status bits are set in the file control block. If it is a valid opcode then the operator code is checked to see if it is valid. If not then it is set to the primary operator code. Next the code is checked to see if it is to the primary operator. If it is then the current operator device code is selected. OPCOUTDV then switches to a routine for the opcode. Each opcode requiring output gets a message buffer to send to the output device. It then copies a message into it and sends it to the device handler. After all operations are finished end is signaled on the file semaphore and OPCOUTDV loops back to wait for another command.

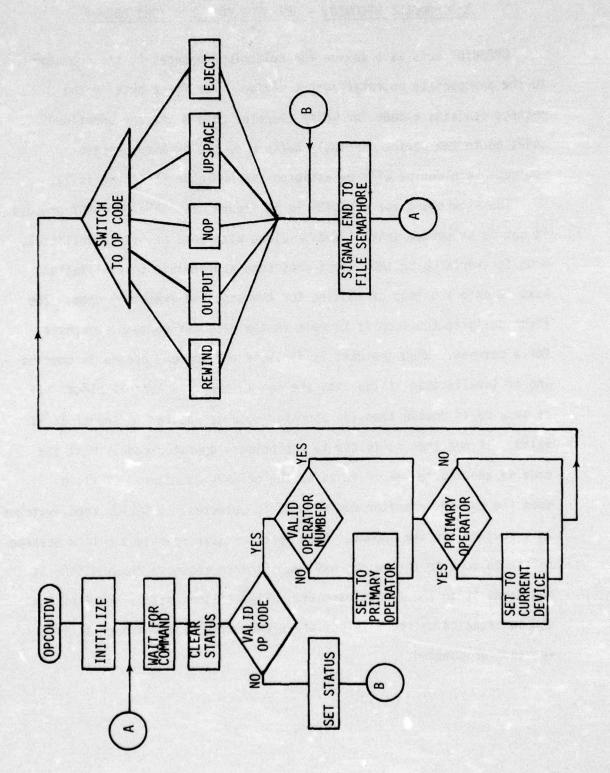


FIGURE 76. FLOWCHART OF OPCOUTDV

## TTYPROS - OPERATOR COMMUNICATIONS TELETYPE DRIVER

TTYPROS is the I/O driver for the teletype, it listens to the keyboard most of the time and regularly checks for output to the teletype. TTYRPOS is an operator communications device driver so it works closely with OPCINDV and OPCOUTDV processes. It runs as a process.

The flowchart for TTYPROS is in figure 77. The process must be set up at system initialization along with some of its capabilities. When TTYPROS starts up it does some appropriate initiliazation, such as setting a flag to start looking for input. TTYPROS goes into a loop alternately looking for input from the keyboard and for any output to the teletype. After a message buffer is obtained from a message buffer pool the teletype is told to start looking for a character. TTYPROS waits for an interval of time. If no input has occured the message queue is checked for any messages. If there is a message the teletype is told to stop looking for any input and start printing the message. After printing, the teletype is told to start looking for input again. If any input is detected TTYPROS reads in a whole line and sends the buffer to the OPCINDV process.

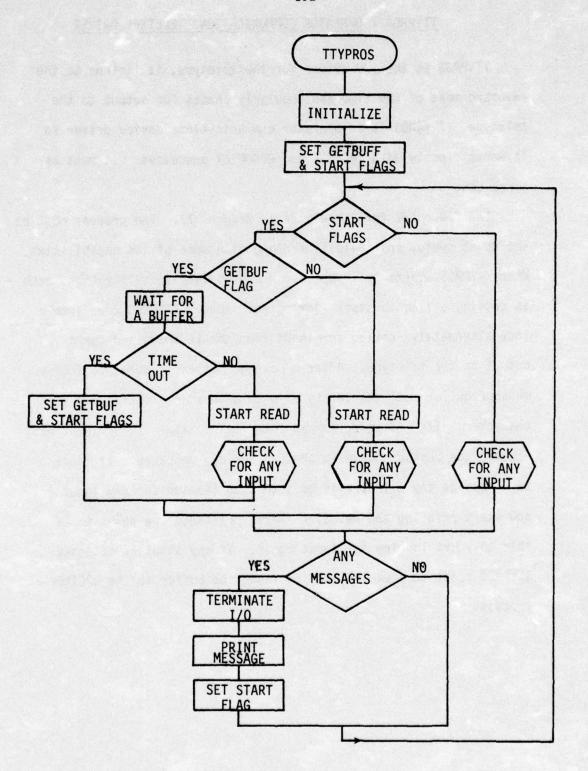


Figure 77. Flowchart of TTYPROS

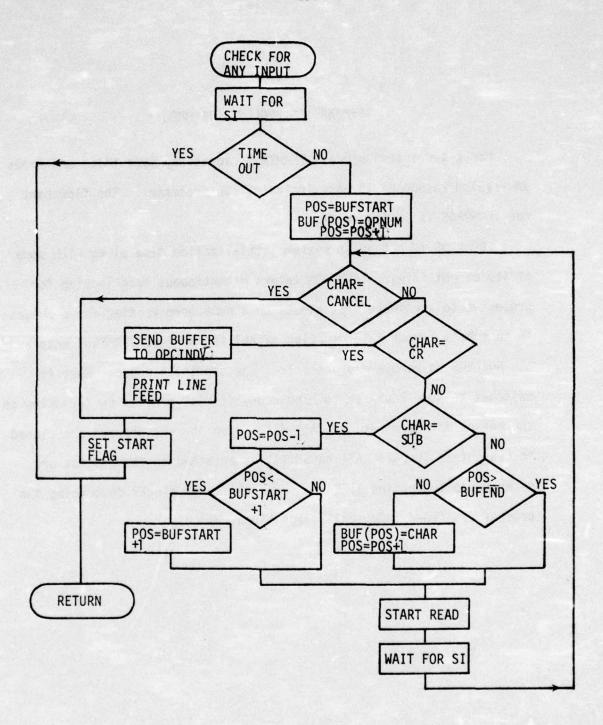


Figure 77. Continued (Internal Subroutine)

#### TERMINATION PROCESS (TERMPROS)

For a terminated process TERMPROS closes any open files and frees any system resources it has, including son processes. The flowchart for TERMPROS is in figure 78.

TERMPROS is set up at system initialization time along with some of its capabilities. TERMPROS enters a continuous loop looking for processes to terminate. It blocks on a semaphore waiting for a process to terminate. When a termination signal is received TERMPROS enters the NUCLEUS to remove a process from the terminate queue. Next it switches to some routines to handle any special processing depending on the reason for terminating. All files open for the process are closed in subroutine TCLOSE. All capabilities attached to the process are detached in subroutine TDACAPS. Last all large blocks describing the process are freed (PCB, call stack block, etc.).

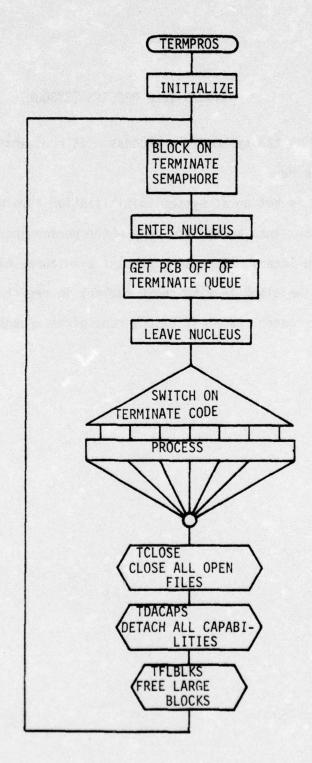


Figure 78. Flowchart of TEMPROS

### SYSTEM IDLE PROCESS (SYSDUM)

SYSDUM is the system idle process. It runs when no other processes are ready to run.

SYSDUM is set up at system initialization time with a priority of zero. It goes into an endless loop where performance evaluation routines may be added later. At present it runs a software clock and compares it to the line clock leaving these numbers in registers so they may be seen on the control panel if SYSDUM runs often enough.

VIII. SOME SYSTEM CONTROL BLOCKS

BLOCK IDENTIFIER (1B)		ENTRY COUNT (4 BITS)	POINTER TO NEXT BLOCK
CAPAPILITY TYPE	P	OINTER TO OBJ	ECT
		DESCRIPTO	?

999 9

20

23 24 27

Figure 79. C-List Block

CAPABILITY INDEX

0w	LINK	PLINK
lw	OLD PSW	RLOLDPSW
2w	UNUSED	
3w	UNUSED	
4w	OLD PROCEDURE	RLOLDPPB
5w	PROTECT	
6w	BITS	
7w		10
8w	REGISTER SAVE AREA	RLREGSA
9w		
Ow		
lw		
2w		
3w		
4w		
5w		

FIGURE 80. CALL PROCEDURE STACK BLOCK

	1B	4 BITS	20 BITS
w	USER COUNT		LINK
w		N	IAME
W		INIT	TAL PSW
w		FLA	G BITS
W		PR	OTECT
W			BITS
N			
N			
N		AD	DRESS
N		LE	NGTH
<b>N</b>			
N			
N			
N			
N			
W			

FIGURE 81. PROCEDURE DESCRIPTOR

W		
w	NAME	
w	COUNTER	
w	QUEUE POINTER	

START I/O MESSAGE SEMAPHORE

DEVICE TABLE ENTRY

)w _	CONTR	ROL DATA
w L	STARTING TRACK	STARTING HEAD SECTOR
w		LENGTH (SECTORS)
w	CURRENT RECO	ORD IN PARTITION

DISC PARTITION TABLE ENTRY

POINTER TO FILE DISC.

MESSAGE SENT TO I/O PROCESS

FIGURE 32. CONTROL BLOCK

OPERATION CODE	F	ILE NAME	
	тс	W	
CONTROL SPECS			
STATUS	*	DEV. STATUS	*
		ACTUAL RECORD LENGTH	*
CURRENT I	RECORD		^
CAP. INDEX FOR BUF MEM		PTR TO FILE DESCRIPTOR	*
DUF MEN		DESURIETUR	

- \* SYSTEM SUPPLIES
- ^ SYSTEM SUPPLIED OR USER SUPPLIED FOR DIRECT ACCESS OPERATIONS

FILE CONTROL BLOCK

Ow	FILE NAME	
1w	CAP. FOR DEVICE SEM.	
2w	PTR TO PARTITION TABLE	

FILE ASSIGNMENT TABLE ENTRY

#### REFERENCE

- 1. SEL <u>Reference Manual Systems 86 Computer</u>, Systems Engineering Laboratories, Ft. Lauderdale, Fla., 1970.
- J. E. Hawkins, Jr., <u>Nucleus for an Experimental Array Radar Operating System</u>, Electrical Engineering Department, Auburn University, Auburn, Alabama, 1974.

# PART TWO

ON THE CORRECTNESS OF
A CLASS OF PROCESS
INTERACTION PRIMITIVES

# Prepared by

W. C. Willis, II Engineering Experiment Station Auburn University Auburn, AL 36830

For

U. S. Army Missile Command Redstone Arsenal, AL 35809

Under Contract
DAAH01-71-C-1303

#### **FOREWORD**

This is a technical summary reporting the progress on contract DAAHO1-71-C-1303 conducted by the Electrical Engineering Department of Auburn University for the U.S. Army Missile Command in Huntsville, Alabama. This report is submitted toward partial fulfillment of said contract.

# TABLE OF CONTENTS

LIST	OF FIGURES
I.	INTRODUCTION
	Objectives Perspectives Outline
II.	MULTIPROGRAMMING SYSTEM MODEL
	Introduction General Model for Programming Systems Computations Model for a Multiprogramming System Additional Definitions Scheduling
III.	EXAMPLE SYSTEM
	Introduction Algorithmic Description of the NUCLEUS Our Model of the NUCLEUS
IV.	CORRECTNESS CONCEPTS
	Introduction Basic Assumptions Sequentiality Producer-Consumer Message Passing Time-out Processing No-block Synchronization Mutual Exclusion
٧.	PROOFS OF CORRECTNESS
	Introduction Proof of Sequentiality Proof of Producer-Consumer Message Passing Proof of Time-out Processing Proof of No-block Synchronization Proof of Mutual Exclusion
VI.	CONCLUSION
	Conclusions Suggestions for Future Pessanch

#### INTRODUCTION

## <u>Objectives</u>

The primary motivation for this work is a desire to develop a formalism that allows statements about the characteristics of multi-programming computer operating systems to be made and proved. In particular, we are interested in formally proving that a system of processes, cooperating by means of certain synchronization and interprocess communications primitives, exhibits the particular operational characteristics that have been informally conjectured for it. To meet these goals, an abstract model of parallel computation is developed. This model allows a greater variety of statements to be made about a system of processes than do other methods.

### Perspectives

The formal basis for the verification of computer programs was first developed by Floyd [1967] and Naur [1966]. Their method states roughly that a program is correct if its execution terminates and the desired output results are produced. A proof of correctness thus consists of two parts: a proof that the program produces the correct output if it halts; and a proof that the program halts. These two proofs are developed by the introduction of at least two logical predicates (or assertations) on the data structure of the program. One assertation represents the constraints on the initial state of the data structure. Another assertation represents the relationships that must hold in the final value of the data vector in

order for the results to be considered correct. Additional assertions may be placed between statements in the program to facilitate the proof. Every loop in the program must be "broken" by an assertion, i.e., an assertion must appear between any two statements of the loop. Then the proof procedure is to show that if the assertion on the input data vector to a section of code is true, then the execution of the code implies that the output assertion is true. Note that the output assertion for one section of code will be the input assertion for the following section (except for the last). Once all of the sections have been verified, the program has been proven correct with respect to the initial and final assertions.

This proof procedure, though it is adequate for sequential programs, is not directly suitable for programs which execute in a parallel manner such as those in real-time systems, process control applications and many operating systems. These programs execute in a cyclic manner and are not designed to halt. They do not produce a single output but rather accept a stream of input values and produce a stream of output values.

The Floyd-Naur procedure described briefly above has been extended by Ashcroft and Manna [1970] to handle parallel programs. In their model, a system of parallel executing programs is represented as a single program with several control paths through it. At each point in time, the (uniprocessor) computer arbitrarily selects one of the paths and executes the instruction determined by the program counter for that path. Then it arbitrarily selects another path, possibly the same, and executes the instruction determined by its program counter. Assertions are placed on each instruction that must represent the state of the data vector when

execution reaches the instruction for each possible path of control to that point. This reduces the problem of proving a parallel program to a problem of proving a sequential type program. However, the number and complexity of the resulting assertions is so great that proofs of practical systems become difficult. Ashcroft and Manna [1970] extend their model by allowing blocks of code that are executed without intervention which helps reduce the number and complexity of the predicates. Levitt [1972] further extends the model by allowing multiprocessor paths and critical sections that allow only one processor at a time to access the section. Even with these enhancements, the method still results in numerous complex assertions that must be proven.

In another extension of the Floyd-Naur formalism, Lauer [1973] develops a flowchart-like notation to represent the behavior of a system of processes. His transition diagrams, similar to flowcharts but with the arcs and nodes reversed, provide a convenient way to describe individual processes. He proves several theorems which show how to develop a transition diagram of a system of processes from the transition diagrams of the individual processes. Floyd-Naur assertions can then be applied to the "product" transition diagram in the same way as for a sequential program for a proof of correctness.

Kahn [1972] presents an approach to proving systems (not necessarily computer operating systems) that involves building a model of the system under study using predefined "building blocks." The Floyd-Naur procedure may then be applied to the model to prove correctness. The building blocks he has defined, while useful in the modeling of systems, are not very

similar to the operations normally encountered in computer systems. For instance, his ADD block, with two inputs and one output operates as follows: "Waits for an input on 1 and an input on 2, both integers; sends the sum of the inputs on 3; deletes the previous set of inputs. Returns to wait." Although computer programs may be modeled by a combination of Kahn's building blocks, the transformation from program to model is non-trivial and difficult to verify.

The approaches described above are all basically similar in that they transform a system of parallel executing processes into a sequential program and then apply the methods of Floyd and Naur. This procedure produces a program with many complex assertions that must be proven.

Until automatic theorem provers are developed, these tedious proofs must be done manually.

A further and perhaps more serious drawback to the methods of Floyd and Naur extended to parallel programs is that some types of correctness criteria are difficult to formulate in the form of predicates on program statements or groups of statements. According to Lauer [1973], the effect of verifying a set of parallel processes with respect to a set of assertions is to prove that "For every execution sequence, the values of the variables have a specific property or relationship." At times we would like to be able to answer questions of the form "Are there any execution sequences with a certain property?" or "Does every execution sequence reach a certain state?" Another drawback to using extensions of Floyd's and Naur's method is that failure to prove an assertion may be due to one of two causes: The assertion may indeed be false; or the theorem prover may simply be unable to prove the assertion.

Gilbert and Chandler [1972] present a method that overcomes some of the shortcomings mentioned above. In their model, the machine may be in one of a number of states. Rules for determining valid transitions between states are derived from the individual processes of the system. Correctness criteria may be stated by listing the "forbidden" states, i.e., states that if ever reached indicate that the system is behaving incorrectly. A mechanical procedure to determine if the system may ever reach a forbidden state is described. For practical systems, the number of states is so large that manual determination of correctness is impractical and automatic methods must be used. This method always results in a positive statement concerning correctness, i.e., "A forbidden state can be reached" or "A forbidden state cannot be reached." Thus, this method overcomes the problem encountered in extensions of Floyd's and Naur's work where failure to find a proof of a predicate does not necessarily imply that a proof does not exist. A drawback of this method, like the previous method, is that some types of correctness criteria are not expressable in terms of forbidden states. Statements that involve the ordering of the sequence of system states cannot be expressed conveniently.

The method presented in this work is based on a model of parallel computation developed by Lipton [1973]. Lipton used his model to compare the "power" of various synchronization primitive systems. The multiprogramming system model developed in this work is essentially the model used by Lipton but generalized to allow modeling of more practical synchronization primitive systems. In addition, its use is different - Lipton used the model to compare systems of processes using different synchronization

primitives; we use it to allow expression and proof of correctness criteria.

### Outline

Chapter 2 develops a model useful for describing the actions and interactions of processes. While developed primarily to model processes in a computer system, the model may be used to describe any type of system of cooperating processes. The model used is an extension of the model used by Lipton [1973]. The basic concepts of multiprogramming are discussed in terms of the model. Various types of process interaction are discussed.

The system that originally motivated this work is described in Chapter 3. First, an algorithmic representation of the nucleus of our example system is presented. Then a description of the system in terms of our multiprogramming system model is given.

Chapter 4 discusses what it means to say that a system is correct. Various criteria that have been previously used in correctness proofs are discussed and represented in terms of our multiprogramming model. Other criteria are also developed and expressed in terms of our model.

Chapter 5 presents proofs of correctness of the system described in Chapter 3. These proofs are based on the correctness criteria discussed in Chapter 4.

Chapter 6 presents our conclusions and suggestions for future research.

### II. MULTIPROGRAMMING SYSTEM MODEL

# Introduction

In this chapter we develop the model for a multiprogramming system.

First a general model for programming systems is introduced. Then additional structure is added to this to produce the model for multiprogramming systems. The model developed is basically the same as the model used by Lipton [1973] for comparing different synchronization primitive systems. However, much of the notation has been changed to conform with nomenclature used in systems with which the author is most familiar. Also, extensions to the model have been made to allow for modeling of message passing facilities.

The model we use is an abstract model, i.e., it abstracts the attributes of computer systems in which we are interested and masks those in which we are not interested. In essence, we are modeling the characteristics of a virtual machine. In practice this is a reasonable thing to do since computer operating systems are frequently implemented as a nucleus of code which creates the virtual machine upon which the processes of the system run.

# General Model for Programming Systems

This section introduces our model of a programming system. It reflects the idea that, in the most basic terms, a computer program consists of a set of instructions operating on a set of data items. We make the

following definitions

 $S = \langle A, D, w \rangle$  is a <u>programming system</u> if A is a set of functions of the form

WHEN B(D) DO D = F(D)

where B is a predicate on D, F is an arbitrary function from D to D, and w is in D. The elements of A are the <u>actions</u> of S.

The set D is the <u>data structure</u> of S. w is the <u>initial data structure value</u> of S. The predicate B(D) is the <u>execution predicate</u> of an action and F(D) is the <u>function</u> of an action.

The data structure D is a composite variable, i.e., the value of an element of D is a composite of the values of all the individual <u>data</u> <u>items</u> in the programming system. Where it is convenient, a reference to the data structure D will be replaced by a reference to the individual data items that comprise D. We will use the representation of D

$$(x_1, x_2, ..., x_k)$$

to denote the composition of D, where each  $x_i$  represents a label that is used to identify the  $i^{th}$  data item of the k data items in D. For example, if D is composed of 4 data items, we might refer to it as

(L,ALPHA, ITEM1, NAME).

We will use the notation

$$(a_1, a_2, ..., a_k)$$

to represent the value of D, where a; is the value of the corresponding

label  $x_i$  in the representation of D. For example, we might find that the 4 element representation of D given above has the value

Also, where convenient, we will refer to a subset of the data items in D using the same notation.

The action predicate of an action B(D) may be denoted in more detail as

$$B(x_1, x_2, ..., x_k).$$

Where B is independent of some  $x_i$ , these may be omitted from the argument list. The assignment D = F(D) of an action expresses k assignments of the form

$$x_1 = f_1(x_1, x_2, ..., x_k);$$
  
 $x_2 = f_2(x_1, x_2, ..., x_k);$   
...  
 $x_k = f_k(x_1, x_2, ..., x_k).$ 

We may shorten this representation when applicable by omitting assignments of the form  $x_i = x_i$ . Also, where  $f_i$  is independent of some  $x_j$ ,  $x_j$  may be omitted from the argument list. There are no restrictions on the form of the  $f_i$ . One particularly useful form is

IF 
$$p(x_1, x_2, ..., x_k)$$
 THEN  $f_{i1}(x_1, x_2, ..., x_k)$   
ELSE  $f_{i2}(x_1, x_2, ..., x_k)$ 

where p is a predicate and  $f_{i1}$  and  $f_{i2}$  are arbitrary functions.

The meaning of the WHEN B(D) DO D = F(D) function form is defined as: at any time while B(D) is satisfied, i.e., is true, the assignment D = F(D) may be made but is not required to be made immediately. Performing the assignment D = F(D) is referred to as executing an action.

Note the following assumptions about the execution of actions:

1. All predicates and functions in an action are evaluated before any assignment to any  $\mathbf{x_i}$  is made. For example, for the subset of D

$$(L,x,y) = (2,4,6)$$

the execution of the action

WHEN L = 2 DO L = 3; 
$$x = 1$$
;  $y = x$ ;

results in the following value of the subset of D

$$(L,x,y) = (3,1,4).$$

2. Actions execute in zero time and no two actions execute at exactly the same instant. This assumption is made to emphasize that our model makes no assumptions about time other than that actions are executed sequentially in time.

Note also that our model is indeterminate in the sense that at each instant only one of possibly many actions whose execution predicates are satisfied is executed and our model makes no restrictions on how this one is chosen.

It is convenient to view a programming system as executable on a virtual processor which operates as follows:

- The actions (program) and w (data) are placed in memory.
- At each instant of time, every action is examined: those whose execution predicates are satisfied are marked ready; those whose execution predicates are not satisfied are marked not-ready.
- Select (in an unspecified manner) one action that is marked ready and execute it.
  - 4. Go to step 2.

Note that the virtual machine will not halt until it no longer has any actions marked ready. Also, to use a programming system to model an actual computer system, actions must be included in the model to simulate the behavior of everything that might affect the computer system, e.g., a command typed in by the operator. Input and output (I/O) operations of a computer system could be modeled by a set of actions which move values between one set of data items (representing the main memory of the computer) and another set of data items (representing the external storage media).

# Computations

A sequence will be denoted in this work by underlined small letters. The empty or null sequence will be denoted by  $\Lambda$ . If  $\underline{a}$  is a finite sequence and  $\underline{b}$  is a finite or infinite sequence, we represent the sequence formed by concatenating  $\underline{a}$  and  $\underline{b}$  as  $\underline{ab}$ . We denote the  $i^{th}$  element of the sequence  $\underline{a}$  by  $\underline{a}_i$ . The number of elements in the sequence  $\underline{a}$  is denoted by LENGTH( $\underline{a}$ ).

We define a <u>computation</u> as a finite or infinite sequence of actions of a programming system. Thus, a computation may be viewed as the sequence of actions executed by the virtual processor as it executes the programming system. Due to the indeterminacy of a programming system, another execution of the same programming system could result in a different computation. In later chapters we will express our correctness criteria in terms of the computations we consider to be correct behavior and those we consider to be incorrect behavior of the programming system. We will introduce notation to make discussion of particular computations more convenient.

Notice that a computation completely describes the behavior of a programming system. The concept of "state of the data structure" used by Gilbert and Chandler [1972] does not. Neither does the concept due to Floyd [1967] and Naur [1966] that "certain relations between the variables of the data structure hold." Both of these methods depend only on the value of the data structure and not on the actions which are executed or the order in which they are executed. Thus, statements involving computations can express a "richer" variety of ideas than can statements based on the state of the data structure.

Although we do not use the concept of "state of the data structure" in expressing our correctness criteria, we nevertheless find the concept useful. Suppose  $S = \langle A, D, w \rangle$  is a programming system. Then we define the function  $STATE(\underline{x})$  whose argument is a computation of  $STATE(\underline{x})$  whose argument is a computation of  $STATE(\underline{x})$  whose value is from the domain of D as follows:

- 1.  $STATE(\Lambda) = W$
- 2. If  $\underline{a}$  is a finite computation of S and f is an action in S, then  $STATE(\underline{a},f) = f[STATE(\underline{a})]$ .

Thus, STATE $(\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_n)$  is the value of D after executing the sequence of actions  $\underline{a}_1\underline{a}_2\ldots\underline{a}_n$ .

To refer to the value of a data item at a particular state of the system, we use the notation  $x_i < Z >$ , where  $x_i$  is the data item and Z is the value of D. For example, the value of COUNTER after executing the computation a is COUNTER<STATE(a)>.

# Model for a Multiprogramming System

In this section we develop our model of a multiprogramming system.

First, we will discuss multiprogramming on an informal level and then present our formal model. The most important concept in multiprogramming is the compact of a process. Although processes have been defined many times by many authors, we will present still another definition expressed in terms of our model of a multiprogramming system. Informally, a process is a program that runs on the virtual machine created by an operating system nucleus of some sort. The nucleus allows several processes to concurrently share the same hardware (either a uniprocessor or a multiprocessor). Each process executes almost as though it alone were executing on a processor. The only exception is that there is a set of primitives (actions in our model of a programming system) that allows processes to interact in a limited way with other processes.

The most frequently encountered primitive is called a <u>synchronization</u> <u>primitive</u> and is used for communication involving <u>events</u> that occur in the system. Typical synchronization primitives allow a process to notify the nucleus that a certain event has occurred or to request that the nucleus not allow the process to execute until a certain event occurs. Thus, these primitives allow processes to synchronize their execution. Note, however, that no information is communicated from one process to another except the fact that the event has occurred.

Another frequently encountered primitive is the <u>interprocess</u> communication primitive. This type of primitive allows data to be sent from one process and received by another process. The data that is sent is called a <u>message</u>. Frequently the interprocess communication primitives will involve synchronization information in the sense that a process may request that the nucleus not allow it to execute until a message is available.

It should be noted that the terms synchronization and interprocess communication have been used with slightly different meanings by other authors.

We say that a process that cannot logically proceed (i.e., make use of a processor) is <u>blocked</u>. A process that is logically able to proceed is termed <u>ready</u>. Each process has its own program counter, which points to its next instruction to be executed. Each action in the system is a part of exactly one process. Previously, we have given an informal description of the operation of a virtual processor upon which a programming system might be executed. Below we present an informal description of the

operation of a virtual processor for a multiprogramming system (as yet not formally defined):

- 1. The actions and initial data structure value are placed in the memory of the virtual machine.
- At each instant of time, every process is examined:
   some will be found to be ready while the remainder will be found to be blocked.
- One of the ready processes will be selected (in an unspecified manner) and the action pointed to by its program counter will be executed.
  - 4. Go to step 2.

Note that our (as yet informal) model of a multiprogramming system partitions the actions of the system into processes and has one program counter per process. Note also that a multiprogramming system is more s structured than a programming system. Since the nucleus of a typical multiprogramming operating system provides the type of virtual processor described above, we feel that our model is an appropriate one.

It is important to note that the action executed in step 3 above may cause one or more processes in the system to become blocked, or it may cause one or more other processes to become ready.

In the implementation of any set of synchronization and interprocess communications facilities, some data structure must be present that allows the nucleus to keep track of the system of processes. It is essential, for correct operation of the system, that operations on this data structure by any process be completed without interference by any other process.

Usually this is accomplished by restricting access to the data structure to the nucleus only, and using an interlock mechanism in the nucleus to prevent simultaneous operations on the data structure. In our model we will allow only synchronization and interprocess communication primitives to access this data structure. Since actions are executed without interruption, we are guaranteed that no harmful interference will occur.

As a process is executing, some of the data upon which it operates will be stored in processor registers. In addition the program counter and some status information will be stored in other registers. When actions from another process are to be executed, the nucleus must first save the contents of all pertinent registers in main memory and then reload the registers with the appropriate register contents of the new process. In our model we mask the saving and restoring of the processor registers by assuming that each process has its own set of registers. As in Dennis and Van Horn [1966], we call data stored in registers private data.

At this point, consider the data structure, D. of a multiprogramming system. Some of the data items are elements of the private data structure of a process (i.e. its registers). We say private since no other process can directly change or examine it. Other data items are in main memory and are accessable only by synchronization and interprocess communication primitives. The remainder of the data is stored in main memory. We restrict access to this last set of data items to operations which are not synchronization or interprocess communication primitives in order to prevent undesired interference between two processes. Figure II-1 illustrates these access restrictions.

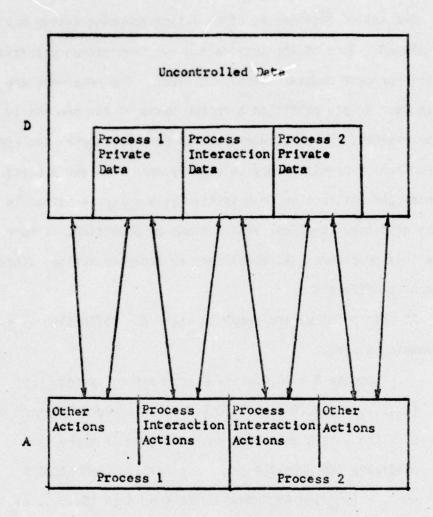


Figure II-1. Data Access Restrictions in a Multiprogramming System.

The set of actions, A, of a multiprogramming system may also be partitioned. Some of the actions are synchronization primitives. Others are interprocess communication primitives. The remainder are neither of the above. We may partition A on the basis of the process in which an action resides. In cases where shared pure procedures are used, a computer instruction or primitive may be used by more than one process. However, we model the instruction or primitive as a <u>separate</u> action in each process. We may also have identical instructions or primitives at more than one place in a procedure. We model them as separate actions since the address of each is different.

At this point we are ready to state our definition of a multiprogramming system.

Suppose  $S = \langle A, D, w \rangle$  is a programming system. Let  $(L_0, L_1, \ldots L_n, P_1, P_2, \ldots, P_n, G, E, M, T)$  be a representation of D. Then S is a multiprogramming system if there is a predicate SYNCHRONIZER on A, a predicate COMMUNICATOR on A, a function PROGRAM-COUNTER from A to  $\{0,1,\ldots,n\}$ , and a function ADDRESS with domain A such that

- 1. If SYNCHRONIZER (f), then there exist a predicate s, and functions  $\mathbf{e}_{\mathrm{E}}$  and  $\mathbf{e}_{\mathrm{p}}$  such that

- b.  $x \neq ADDRESS(f)$ .
- 2. If COMMUNICATOR(f), then there exist a predicate c and functions  $m_{M}$  and  $m_{P}$  such that
  - a. f is of the form

WHEN 
$$(L_i = ADDRESS(f))Ac(M,T,P_i)$$
 DO  
 $L_i = y;$   
 $M = m_M(M,P_i);$   
 $P_i = m_p(M);$ 

where i = PROGRAMMER-COUNTER(f).

- b.  $y \neq ADDRESS(f)$ .
- 3. If not SYNCHRONIZER(f)  $\Lambda$  not COMMUNICATOR(f), then there exist functions b,  $t_G$  and  $t_P$  such that
  - a. f is of the form

WHEN 
$$L_i$$
 = ADDRESS(f) DO  

$$L_i = b(G, P_i, L_i);$$

$$G = t_G(G, P_i);$$

$$P_i = t_P(G, P_i);$$

where i = PROGRAM-COUNTER(f).

- b. for all x in D,  $b(c) \neq ADDRESS(f)$ .
- 4. If ADDRESS(f) = ADDRESS(g) and PROGRAM-COUNTER(f) = PROGRAM-COUNTER(g), then f = g.
  - 5. There is one special action in A which is exactly WHEN  $L_0$  = 1 D0 T = T + 1;  $L_0$  = 1;

The data structure of a multiprogramming system consists of the following:

- 1. Program counters. These data items are labelled  $L_{\hat{i}}$ , for i = 0 to n, where n is the number of processes in the system (program counter  $L_{\hat{0}}$  has a special use described below). There is one program counter per process. Each program counter points to the next action from its process to be executed. The value of the program counter is changed by the execution of the action. No process can access the program counter of another process. Program counters have been separated from the private data simply for convenience; they are referred to frequently. Note that the program counter of an action can be readily determined by examining the execution predicate of the action. By our definition of the form of all allowable actions in a multiprogramming system, the execution predicate of each action refers explicitly to exactly one program counter.
- 2. Private data structure. These are composite data items and are labelled  $P_i$  for i=1 to n. Each process contains its own private data, i.e., general purpose registers, status registers, etc. (Program counters are not considered private data.) No process can access another process's private data. Private data  $P_i$  can be accessed by any action f such that

PROGRAM-COUNTER(f) = i.

3. Global data structure. This is a composite data item and is labelled G. Global data items may be accessed by any action f in A such that

(not SYNCHRONIZER(f) A not COMMUNICATOR(f)) = true.

- 4. Synchronization data structure. This composite data item, labelled E, consists of the data structure necessary to implement the synchronization primitives of the multiprogramming system. A data item in E may be accessed by an action f only if SYNCHRONIZER(f) is true.
- 5. Interprocess communication data structure. This composite data item, labelled M, consists of the data structure necessary to implement the interprocess communication primitives of the multiprogramming system. A data item in M may be accessed by an action f only if COMMUNICATOR(f) is true.
- 6. System clock. This data item, labelled T, is incremented by a special action. It simulates a timer and may be read by SYNCHRONIZER and COMMUNICATOR actions.

The predicate SYNCHRONIZER(f) is true if f is of the form described for a synchronization primitive. The predicate COMMUNICATOR(f) is true if f is of the form described for an interprocess communication primitive. The function PROGRAM-COUNTER(f) yields the index of the program counter used by f. Since there is a one-to-one correspondence between program counters and processes, this index may also be viewed as a process identifier. The function ADDRESS(f) yields the address of the action f in the computer's memory.

We see from condition 1 in the definition of a multiprogramming system that a SYNCHRONIZER action may be executed only when its program counter points to it and the predicate s is true.

The program counter of the process must be changed by the execution of the SYNCHRONIZER actions. In a typical operating system, the new value of the program counter will be the address of the instruction following the synchronization primitive. Only T,  $L_i$  and data items in E and  $P_i$  may be examined. Only  $L_i$  and data items in E and  $P_i$  may be changed. Note that  $e_E$  and  $e_p$  are functions of E alone.

From condition 2 we see that a COMMUNICATOR action may be executed only when its program counter points to it and the predicate c is true. The program counter of the process must be changed. Only T,  $L_i$  and data items in M and  $P_i$  may be examined. Only  $L_i$  and data items in M and  $P_i$  may be changed. Note that  $m_M$  is a function of both M and  $P_i$ , while  $m_p$  is a function of M alone.

Note that every action must change its program counter. This requirement reflects reality by ruling out the possibility of an action which truly does nothing, and thus may "hang up" the machine in some sort of do-nothing loop.

The triple  $(s,e_E,e_p)$  completely characterizes a synchronization primitive. The set formed by the union of all unique triples from all

SYNCHRONIZER actions in a multiprogramming system is called a <u>synchronization primitive system</u>. Similarly, the set formed by the union of all unique triples of the form  $(c,m_M,m_p)$  from all COMMUNICATOR actions is called an interprocess communication primitive system.

Condition 4 states that two actions are equal if they have the same address and the same program counter.

Condition 5 defines the process which is responsible for updating the system time T. It consists of only one action whose primary function is to add 1 to T. It is an action which is always ready to be executed. T may be examined by SYNCHRONIZER and COMMUNICATOR actions. Taken together, the process defined by condition 5 and the data item T constitute what is commonly called a real-time clock feature of a digital computer. It is basically a counter that is periodically incremented by the hardware and can be accessed by the programs in the machine. Although it is a part of every multiprogramming system, we will frequently omit the action of the "timer" process from our description of a multiprogramming system for simplicity.

We now state the differences between our model for a multiprogramming system and the model used by Lipton [1973]. His model contains no interprocess communication actions; he allows synchronization primitives as the only means by which processes may interact. Many operating systems do provide interprocess communication facilities, e.g. Organick [1972], so this type of primitive should be included in our model. His model does not include the concept of private data, a feature which is useful where the

nucleus returns status or messages in registers or takes messages from registers.

A significant but more subtle difference between the two definitions is due to the difference in use of the two models. Lipton includes an additional condition in his model which, when converted into our notation and extended to include interprocess communication primitives, states the additional condition:

6. If not SYNCHRONIZER(f) Anot COMMUNICATOR(f), PROGRAM-COUNTER(f) ≠ PROGRAM-COUNTER(g), and a is a finite computation then

 $f(g(STATE(\underline{a}))) = g(f(STATE(\underline{a}))).$ 

He then shows that this condition is equivalent to the statement "if f and g share a variable, then they are enclosed in critical sections."

As we shall see in Chapter IV, this statement is one criteria of a correct multiprogramming system. If we examine the restrictions placed on the actions of a multiprogramming system as defined by Lipton, we see that there are basically two types of restrictions. One type deals with restrictions on what actions are allowed (and enforced) by the nucleus. These are basically a matter of whether or not a particular action may access aparticular data item. The other form of restriction placed on the actions deals with limiting the access of a process to a data item to particular segments of time and forbidding access at all other times. Enforcement of this type of restriction lies with the programmer at the time he writes the procedures that are executed by the process. Only he has the necessary intimate knowledge of his processes' interaction. The

operating system cannot determine on its own whether or not the interaction is as desired by the programmer.

For Lipton's purposes, i.e. comparing different synchronization primitive systems, the inclusion of condition 6 is appropriate. It merely requires that cooperating processes conform to certain forms which will insure mutual exclusion during execution of critical sections. For our purposes, i.e. modeling multiprogramming systems so that conclusions about the suitability of synchronization and interprocess communication primitives can be made, the correctness criteria expressed by condition 6 should not be built into the model. We prefer to let our model include only information about the operation of the synchronization and interprocess communication primitives and leave statements about the operation of a system of processes for proof.

# Additional Definitions

We now introduce several notions that will facilitate discussion of multiprogramming systems. They allow us to express most ideas about systems of processes while suppressing unnecessary or uninteresting detail.

The notion of "process" has been used in an informal way in previous sections. Now we formalize it. Suppose  $S = \langle A, D, w \rangle$  is a multiprogramming system. For actions f and g, the predicate PROCESS (f,g) is true if PROGRAM-COUNTER(f) = PROGRAM-COUNTER(g). The notion of process is an equivalence relation on the actions of S. We let PROCESS-i denote the i<sup>th</sup> equivalence class of the relation PROCESS, i.e.

PROCESS- $i = \{f | PROGRAM-COUNTER(f) = i\}$ . This formalizes our earlier idea that there is a one-to-one correspondence between processes and program counters.

Sometimes it is useful to refer to that set of actions of a multiprogramming system which are pointed to by the set of program counters. We formalize this notion in the following definition:

POINTER-SET(
$$\underline{a}$$
) = {f|L<sub>i</sub>\underline{a})> = ADDRESS(f)},

where  $\underline{a}$  is a finite computation and  $\underline{i}$  = PROGRAM-COUNTER(f). We wee that since  $\underline{L}_{\underline{i}}$  may point only to those actions in PROCESS-i, POINTER-SET( $\underline{a}$ ) contains exactly one action from each process.

At each instant, the execution predicate of an action in POINTER-SET  $(\underline{a})$  may or may not be satisfied. We make the following definitions to distinguish between the two cases:

READY-SET(
$$\underline{a}$$
) = {f|f[STATE( $\underline{a}$ )]  $\neq$  STATE( $\underline{a}$ )},

and

$$BLOCKED-SET(a) = POINTER-SET(a) - READY-SET(a)$$
,

where  $\underline{a}$  is a finite computation and "-" is set subtraction. Informally, we say that after executing the finite computation  $\underline{a}$ , a process which has an action in READY-SET( $\underline{a}$ ) is  $\underline{ready}$  and a process which has an action in BLOCKED-SET( $\underline{a}$ ) is  $\underline{blocked}$ .

# Scheduling

Our model of a multiprogramming system has no built-in assumptions about the selection of the next action to be executed from the set of ready processes. In an operating system, this function is generally referred to as process scheduling or simply <u>scheduling</u>. In this section we discuss several aspects of scheduling, especially the way to represent scheduling in terms of our model.

Consider the set of all possible computations of a multiprogramming system. Obviously some of this set will conform to whatever scheduling policy is implemented by our operating system while others of the set will not. We may distinguish between the two subsets by defining a predicate on the computations of the multiprogramming system. Then we consider only those computations that satisfy the predicate. Suppose S is a multiprogramming system. Then R is a scheduler for S if R is a predicate on the computations of S.

We use a scheduler to select those computations of a multiprogramming system that obey the implemented scheduling policy. For instance, we may use a scheduler to enforce a priority type of scheduling so that the highest priority process of the set of ready processes is chosen for execution. We may choose a scheduler to enforce a fairness policy. For example, in a time-sharing system, processes are allocated a time-slice to run and then are not allowed to run while the remaining processes get their turn. Other types of schedulers are also possible.

#### III. EXAMPLE SYSTEM

### Introduction

This chapter presents the nucleus of a multiprogramming operating system which will be proven to be correct in Chapter V according to the correctness criteria discussed in Chapter IV. This nucleus is part of the operating system described in Hawkins [1974]. We will refer to the nucleus of ROS as the NUCLEUS for the remainder of this work. The NUCLEUS was written in assembly language, but is described in this chapter in PL/I. This allows us to suppress unnecessary detail and to describe the system without having to discuss the architecture of the particular machine upon which the NUCLEUS is implemented. Only process interaction primitives and their associated data bases will be discussed, since we are only interested in proving statements about process interaction. Then we will map the process interaction primitives of the NUCLEUS into actions in our multiprogramming system model.

# Algorithmic Description of the NUCLEUS

The primitives of the NUCLEUS are accessed by executing a Call Monitor (CALM) instruction, which generates an interrupt and passes a 10-bit code to the interrupt handler. The interrupt handler decodes the 10-bit code to determine the type of service requested by the calling process and calls the appropriate subroutine to perform the requested function. Only the six of these services that are directly related to process interaction will be described below.

The data structure of the NUCLEUS used by the process interaction primitives consists of the following:

- 1. Process Control Blocks (PCB's). There is one PCB per process. The private data of a process is stored in its PCB whenever the process is not actually running on the processor. When the process is running, some of its private data is stored in the machine registers while the remainder is stored in the PCB. In addition to private data, PCB's also contain links used to chain the PCB's on the various linked lists of the NUCLEUS and other data not related to process interaction.
- 2. Ready Process Queue (RPQ). When a process is logically able to proceed (i.e. use the processor), it is placed on the RPQ. Processes on the RPQ are ordered by effective priority, which is a function of the priority stored in a process's PCB and the length of time the process has been on the RPQ. When the NUCLEUS ceases execution of one process and needs a new process for execution, it removes the first process (highest effective priority) from the RPQ. The selected process becomes the current process (CP).
- 3. Event Channels (EC's). EC's are structured data items operated on by the synchronization primitives. An EC consist of two data items: a counter and a queue. While a process is blocked upon an EC, its PCB is placed on the queue of the EC. The counter of the EC contains information about either the number of processes blocked on the EC or the number of times that processes will not become blocked on the EC.

- 4. Message Queues (MQ's). MQ's are structured data items operated on by the interprocess communication primitives. An MQ consists of two data items" a counter and a queue. While a process is waiting on a message at an MQ, its PCB is placed on the queue of the MQ. While messages are waiting at an MQ for a process to receive them, they are stored on the queue of the MQ. The counter of the MQ contains information about either the number of processes waiting on a message or the number of messages waiting for a process.
- 5. Timer Chain (TC). The timer chain is a linked list of processes which are also on the queue of some EC or MQ. At the time a process becomes blocked on an EC or MQ, the process may specify the maximum period of time it should be allowed to remain blocked. If the specified time interval expires before the process is otherwise unblocked from the EC or MQ, the process will be removed from the queue and the timer chain and placed on the RPQ.
- 6. TIMER. This data item is initialized to zero when the NUCLEUS is initialized and is incremented every 1/60 of a second after that. TIMER is used to determine if a process has timed-out.
- 7. Message buffers. These are 4-word blocks allocated from a pool of 4-word blocks in the NUCLEUS. They are used to store messages until a process receives them.

Figure III-1 shows the data declarations that are global to the procedures described later in this section. The NUCLEUS is written in assembly language, but we describe it in PL/I. A higher level language is used in order to mask irrelevant details of the machine architecture.

```
MAXIMUM_WAIT_TIME FIXED BINARY(31),
                                                                                                                                                                                                                                                                                                                                                                TIMER_CHAIN_LINK FOINTER,
TIME_OUT_TIME FIXED BINARY(31),
BLOCKED_SEMAPHORE_PTR POINTER;
                                                                                                                                                                                                          LINK_INFO LIKE LIST_ELEMENT,
                                                                                                                                                                                                                                                                   LINK_INFO LIKE LIST_ELEMENT,
                                                                                                                                                                                                                                                                                      SEMAPHORE_POINTER POINTER.
                                                                                           EC BASED(X) LIKE SEMAPHORE; MQ BASED(X) LIKE SEMAPHORE;
                                                                                                                                                                       PRIORITY FIXED BINARY(7);
                                                      COUNTER FIXED BINARY(31),
                                                                                                                                                                                        MESSAGE_BUFFER BASED(X),
                (CPP, RPOP, TCP, X) POINTER:
                                                                                                                                LIST_ELEMENT BASED(X),
TIMER FIXED BINARY (31);
                                  1 SEMAPHORE BASEDIX),
                                                                                                                                                                                                                                                                                                                            MESSAGE BIT (64),
                                                                                                                                                                                                                                                                                                                                            STATUS BIT(1),
                                                                                                                                                   LINK POINTER,
                                                                                                                                                                                                                             800Y BIT(64);
                                                                         GUEUE POINTER;
                                                                                                                                                                                                                                                 PCB BASED(X).
                                                                                                                                                                                           DECLARE
DECLARE
                                                                                                                                                                                                                                                  DECLARE
                 DECLARE
                                    CFCLARE
                                                                                                               DECLARE
                                                                                                                                 DECLARE
                                                                                            DECLARE
```

Figure III-1. NUCLEUS data base declarations.

PL/I was chosen because of its wide use and its ease in describing the use of pointers. The data items described are used to implement the data structure presented earlier. The NUCLEUS data base contains other data items which are irrelevant to the process interaction facilities, and so are omitted.

TIMER is a 32-bit word that is incremented at a 60 Hz rate. CPP points to the PCB of the current process. RPQP points to the Ready Process QUEUE, a linked list of PCB's. The PCB's are ordered on their priority, the higher priority nearest the head of the list. When a process is added to the RPO, it is placed after all processes having the same or equal priority. Thus, within a priority level, the processes are ordered on their arrival times. Processes are removed from the front of the RPQ when the NUCLEUS needs a new process to run. TCP points to the timer chain, a linked list of PCB's that are blocked on semaphores' queues (see below). When a process becomes blocked, the process may specify the maximum length of time it is willing to wait to be unblocked. If it is not unblocked by another process, the NUCLEUS unblocks it and returns a status flag indicating that the process timed-out. When a call is made to the NUCLEUS for a service that could result in the process becoming blocked, the maximum-wait-time specified is checked. If it is zero, the process is not put on the timer chain. If it is not zero, the time-out-time for the process is computed and put in the PCB. The process is then put on the timer chain. Processes on the timer chain are ordered on their time-out-time, the earliest time-out-times being nearest the head of the list. Each time TIMER is updated, the processes on the timer chain whose time-out-times have occurred are removed and unblocked. Only

processes at the head of the timer chain need be examined. X is a dummy variable used because PL/I requires that declarations of BASED variables specify a pointer.

The synchronization and interprocess communication data bases consist of <u>semaphores</u>, i.e. structured data items consisting of a counter and a queue pointer. Semaphores used in the synchronization data base are called event channels. Semaphores used in the interprocess communications data base are called message queues. The structure of semaphores is declared and the structure of EC's and MQ's is declared to be the same as that of semaphores. Semaphores are allocated by the NUCLEUS at the request of processes. The declarations in Figure III-1 merely give their format.

The same type of linked list structure is used in various parts of the NUCLEUS to chain different types of list elements. To enable list processing subroutines to handle any type of element with the same linking conventions, we declare a structured variable called a LIST-ELEMENT. It consists of a link and a priority. This merely gives the format of the standard linking convention; no LIST-ELEMENTs are ever allocated.

Message buffers are used to buffer messages until a process is ready to receive them. They conform to the standard linking conventions. The messages passed are 64-bit strings. Message buffers are allocated when necessary by the NUCLEUS.

PCB's contain information private to a process in addition to various link words. A PCB is allocated when a process is created and exists for the life of the process. PCB's conform to the standard linking conventions.

Before calling the NUCLEUS for a system service, a process loads particular registers with appropriate parameters. Immediately upon entering the NUCLEUS, the registers of the process are stored in its PCB. Also, when returning control to the process, the registers are reloaded from the PCB. During the time the NUCLEUS is executing, the process's registers are stored in its PCB. Hence, without loss of essential detail, we represent the arguments by name in the PCB rather than by register number. Information returned to the process is named, also. SEMAPHORE-POINTER points to the EC or MQ involved in the request for a NUCLEUS service. (Actually pointers are not passed to the NUCLEUS, only indices into a table of pointers which only the NUCLEUS may access. Since there is a one-to-one correspondence between indices and pointers, for simplicity we show pointers being passed. There is no loss of essential detail.) MAXIMUM-WAIT-TIME is passed on certain NUCLEUS calls and is used in connection with the timer chain. A MESSAGE consists of two 32-bit words either passed to the NUCLEUS or returned from it. STATUS is returned to the process after NUCLEUS calls to inform the process of special conditions. It is one bit (0 or 1) and is placed in the condition code register to allow easy testing. TIMER-CHAIN-LINK is used to link the process on the timer chain. TIME-OUT-TIME is the time at which the process should be timed-out. It has no significance unless the process is on the timer chain. BLOCKED-SEMAPHORE-PTR points to the semaphore upon which the blocked process is waiting. It only has significance when the process is on the timer chain. It is used when a process times-out and must be removed from a semaphore's queue.

At this point we begin our description of the relevant procedures of the NUCLEUS. First, the environment in which they execute will be described. Then a description of each procedure will be given.

When the NUCLEUS is entered, all lower priority interrupts are disabled so that the NUCLEUS cannot be interrupted by any lower priority interrupt. In addition, all higher level interrupts are mapped down below the NUCLEUS level so that they cannot interrupt the NUCLEUS. Thus, all processing in the NUCLEUS is non-interruptable. Therefore, there is no problem of undesired interference between the various procedures of the NUCLEUS. Processes pass parameters to the NUCLEUS in their registers. Upon entry to the NUCLEUS, the registers are saved in the process's PCB. NUCLEUS procedures must extract the parameters from the PCB to use them. Also, when parameters are to be returned to a process, the NUCLEUS procedures must store the parameters in the PCB. Just prior to exit from the NUCLEUS, the registers are reloaded from the PCB. In our description of the NUCLEUS, we use the name of the parameter rather than the number of the register containing it.

An important data item, manipulated by all of the process interaction primitives, is the semaphore. The semaphore implemented by the NUCLEUS is similar to the general semaphore of Dijkstra [1968]. It is a mechanism which allows a process to proceed or to be blocked depending on the value of a data item, namely the counter of the semaphore. Dijkstra defines two operations which may be performed on a semaphore: a P operation or a V operation, denoted by P(semaphore) and V(semaphore). These are the only operations allowed on a semaphore. The counter of the semaphore is

initialized to zero. A P(semaphore) decrements the counter of the semaphore by 1. If the resulting value of the counter is non-negative, control is returned to the calling process. If the resulting value is negative, the process is placed on the queue of the semaphore and a new process is selected for execution. In the latter situation, the process is said to have become blocked. A V(semaphore) increments the counter of the semaphore by 1. If the resulting value of the counter of the semaphore is greater than 0, control is returned to the calling process. If the result is 0 or negative, a process is removed from the queue of the semaphore and added to the set of ready processes, from which a process is selected for execution. Thus, a P(semaphore) represents a potential blocking of the calling process and a V(semaphore) represents a potential unblocking of a different process.

The process interaction primitives implemented by the NUCLEUS are extensions of Dijkstra's P and V operations. There are two types of semaphores: event channels and message queues. (Message queues are further from Dijkstra's definition of a semaphore than event channels since messages may be queued up on a message queue.) Each of the two types may be operated upon by 3 primitives. In addition, the real-time clock interrupt handler may manipulate both types of semaphores.

The primitives that may operate upon event channels are:

 SIGNAL (event-channel). This is essentially the same as Dijkstra's V operation. It is used by a process to inform the NUCLEUS that an event has occurred.

- 2. BLOCK (event-channel, maximum-wait-time). This is similar to Dijkstra's P operation, but with the additional feature that a process may indicate the maximum amount of time that it is willing to be blocked. If it is not unblocked by some other process doing a SIGNAL on the same event channel within the specified amount of time, the NUCLEUS will unblock the process and return an indication that it timed-out. BLOCK is used by a process to indicate to the NUCLEUS that it cannot logically proceed until the indicated event occurs.
- 3. CHECK (event-channel). This primitive performs a BLOCK operation if the process will not be blocked by doing it. Otherwise it returns status indicating the fact that the process would have become blocked had a BLOCK operation actually been done. CHECK may be used by a process in forcing an event channel to a known state.

The primitives that operate upon message queues are similar to those that operate upon event channels, except that instead of involving the communication of information that an event has occurred, they allow 64-bit messages to be communicated between processes. The implemented primitives are:

1. SEND (message-queue, message). This primitive sends the indicated 64-bit message to the message queue indicated. Its operation is similar to a V operation. The counter of the semaphore is incremented by 1. If the result is greater than zero, a message buffer is allocated, the message copied into it, and it is put on the queue of the message queue. Then control is returned to the

calling process. If the result is 0 or negative, a process is removed from the queue of the message queue, the message is copied from the calling process to the newly unblocked process, and the latter is added to the ready process queue. The counter of the semaphore then indicates, if positive, the number of messages waiting for a process to receive them, and, if negative, the number of processes blocked on the message queue awaiting messages.

- 2. WAIT (message-queue, message, maximum-wait-time). This primitive allows a process to receive a message from a message queue if there is one waiting, or to be blocked until one is sent if there is not one waiting. Its operation is similar to a P operation. The counter of the message queue is decremented by 1. If the result is non-negative, a message buffer is removed from the queue of the message queue, the message is copied from the message buffer to the calling process and control is returned to the process. If the result is negative, the process is placed on the queue of the message queue, and a new process is selected to run. Like the BLOCK primitive, the process may specify the maximum amount of time it is willing to be blocked.
- 3. RECEIVE (message-queue, message). This primitive is basically a conditional WAIT. If a message is available at the specified message queue, then RECEIVE has the effect of a WAIT primitive. If there is no message, i.e. the counter of the message queue is not greater than zero, control is returned to the calling process with status information indicating that no message was waiting.

```
CALL DEQUEUE(X,P->EC.QUEUE);
IF X->PCB.TIMER_CHAIN_LINK ¬= NULL THEN CALL UNLINK_TC(X);
IF X->PCB.PRIORITY > CPP->PCB.PRIORITY
                                                            P = CPP->PCB.SEMAPHORE_POINTER;
P->EC.COUNTER = P->EC.COUNTER + 1;
                                                                                                                                                                                                                               CALL ENQUEUE (CPP, RPQP);
                                                                                                                                                                                                                                                                                                                 CALL ENQUEUE(X,RPOP);
                                      CPP->PCB.STATUS = "0"E;
                DECLARE (P,X) POINTER;
                                                                                                  IF P->EC.COUNTER <= 0
SIGNAL: PROCEDURE:
                                                                                                                                                                                                                                                   CPP = X;
                                                                                                                                                                                                                                                                                             ELSE DO;
                                                                                                                                                                                                                                                                        END:
                                                                                                                                                                                                                                                                                                                                     END:
                                                                                                                                                                                                                                                                                                                                                                              END SIGNAL;
                                                                                                                       THEN DO:
```

Figure III-2. The procedure SIGNAL.

0

Parents Parents

IF CPP->PCB.MAXIMUM\_WAIT\_TIME > 0 THEN CALL LINK\_TC(CPP,P); CALL ENQUEUE(CPP,P->EC.QUEUE); CALL DEQUEUE(CPP,RPQP); P = CPP->PCB.SEMAPHORE\_POINTER: P->EC.COUNTER = P->EC.COUNTER - 1: CPP->PCB.STATUS = "6"B; IF P->EC.COUNTER < 0 DECLARE P POINTER; THEN DO:

END BLOCK;

BLOCK: PROCEDURE;

```
CHECK: PROCEDURE;

DECLARE P POINTER;

CPP->PCB.STATUS = *0*B;

P = CPP->PCB.SEMAPHORE_POINTER;

IF P->EC.COUNTER > 0

THEN P->EC.COUNTER = P->EC.COUNTER - 1;

ELSE CPP->PCB.STATUS = *1*B;

END CHECK;
```

Figure III-4. The procedure CHECK.

```
IF X->PCB.TIMER_CHAIN_LINK -= NULL THEN CALL UNLINK_TC(X);
IF CPP->PCB.PRIORITY > X->PCB.PRIORITY
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    X->MESSAGE_BUFFER.PRICRITY = CPP->PCB.PRICRITY;
CALL ENQUEUE(X,P->MQ.QUEUE);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                ALLOCATE MESSAGE_BUFFER SET(X);
X->MESSAGE_BUFFER.BODY = CPP->PCB.MESSAGE;
                                                                                                                                                                                                              X->PCB.MESSAGE = CPP->PCB.MESSAGE;
                                                                          P = CPP->PCB.SEMAPHORE_POINTER;
P->MO.COUNTER = P->MQ.COUNTER + 1;
                                                                                                                                                                                    CALL DEQUEUE (X,P->MQ.QUEUE);
                                                                                                                                                                                                                                                                                            THEN CALL ENGUEUE(X, RPQP);
                                                                                                                                                                                                                                                                                                                                                CALL ENQUEUE (CPP, RPOP);
                                                   CPP->PCB.STATUS = '0'8;
                         DECLARE (P,X) POINTER;
                                                                                                                                 IF P->MO.COUNTER < 1
                                                                                                                                                                                                                                                                                                                                                                           CPP = X;
                                                                                                                                                                                                                                                                                                                      ELSE DO:
SEND: PROCEDURE;
                                                                                                                                                                                                                                                                                                                                                                                                                                                        FLSE DO:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  END SEND:
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           END:
```

Figure III-5. The procedure SEND.

```
RECEIVE: PROCEDURE;

DECLARE (P,X) POINTER;

CPP->PCB.STATUS = 0.0B;

P = CPP->PCB.SEMAPHORE_POINTER;

IF P->MQ.COUNTER > 0

THEN DO;

CALL DEQUEUE(X,P->MQ.QUEUE);

P->MQ.COUNTER = P->MQ.COUNTER - I;

CAP->PCB.MESSAGE = X->MESSAGE_BUFFER;

END;

ELSE DO;

CPP->PCB.MESSAGE = (64) 00 B;

CPP->PCB.MESSAGE = (64) 00 B;

CPP->PCB.MESSAGE = 10 B;

END;

END;

END;

END;

END;

END RECEIVE;
```

Figure III-6. The procedure RECEIVE.

```
IF CPP->PCB.MAXIMUM_WAIT_TIME > 0 THEN CALL LINK_TC(CPP,P);
CALL ENGUEUE(CPP,P->MQ.QUEUE);
CALL DEQUEUE(CPP,RPQP);
                                                                                                                                                                                                                                               CPP->PCB.MESSAGE = MESSAGE_BUFFER.BUDY;
                                        P = CPP->PCB.SEMAPHORE_POINTER;
P->MQ.COUNTER = P->MQ.COUNTER - 1;
                                                                                                                                                                                                                               CALL DEGUEUE(X,P->MQ.QUEUE);
                                                                                                                                                                                                                                                                      FREE X->MESSAGE_BUFFER;
DECLARE (P,X) POINTER;
CPP->PCB.STATUS = *0*8;
                                                                                 IF P->MQ.COUNTER < 0
                                                                                                        THEN DO:
                                                                                                                                                                                                          ELSE DO:
                                                                                                                                                                                                                                                                                                              WAIT;
                                                                                                                                                                                     END:
                                                                                                                                                                                                                                                                                                              END
```

WAIT: PROCEDURE;

```
TCP = TCP->PCB.TIMER_CHAIN_LINK;
X = P->PCB.BLOCKED_SEMAPHORE_PTR;
X->SEMAPHORE.COUNTER = X->SEMAPHORE.COUNTER + 1;
                                                                                            IF TCP->PCB.TIME_OUT_TIME > TIMER THEN RETURN;
                                                                                                                                                                                                                                                                                                                                                                                                                                        IF P->PCB.PRIORITY < CPP->PCB.PRIORITY
                                                                                                                                                                                                                                                                                                                                                                   Y->LIST_ELEMENT.LINK = P->PCB.LINK;
P->PCB.TIMER_CHAIN_LINK = NULL;
P->PCB.STATUS = '1'B;
                                                                      LOOP: IF TCP = NULL THEN RETURN;
                                                                                                                                                                                                                    Y = ADDR (X->SEMAPHORE.QUEUE);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                   THEN CALL ENQUEUE (P, RPCP);
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 CALL ENQUEUE (CPP, RPQP);
                                                                                                                                                                                                                                                                                                                     Z = Z->LIST_ELEMENT.LINK;
                    DECLARE (P,X,Y,Z) POINTER;
                                                                                                                                                                                                                                            Z=X->SEMAPHORE.QUEUE;
                                                TIMER = TIMER + 1;
                                                                                                                                                                                                                                                                      DO WHILE (Z == P);
RTCINT: PROCEDURE;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           CPP = P;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   END RTCINT;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         GOTO LOOP;
                                                                                                                                                                                                                                                                                            : Z = X
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      END:
                                                                                                                      P = TCP;
                                                                                                                                                                                                                                                                                                                                              END:
```

Figure III-8. The procedure RTCINT.

```
ENQUEUE: PROCEDURE(P,0);

DECLARE (P,0,x,Y) POINTER;

X = ADDR(Q);

Y = Q;

LOOP: IF Y = NULL THEN GOTO FINI;

IF Y->LIST_ELEMENT.PRIORITY < P->LIST_ELEMENT.PRIORITY

THEN GOTO FINI;

X = Y;

Y = Y->LIST_ELEMENT.LINK;

GOTO LOOP;

FINI: P->LIST_ELEMENT.LINK = Y;

X->LIST_ELEMENT.LINK = P;

END ENOUGUE;
```

Processed and Pr

DEQUEUE: PROCEDURE(P,Q);

DECLARE (P,Q) POINTER;

P = Q;

Q = Q->LIST\_ELEMENT.LINK;

END DEQUEUE;

```
LOOP: IF Y = NULL THEN GOTO FINI;

IF P->PCB.TIME_OUT_TIME < Y->PCB.TIME_OUT_TIME THEN GOTO FINI;

X = ADDR(Y->PCB.TIMER_CHAIN_LINK);
                                                     P->PCB.TIME_OUT_TIME = P->PCB.MAXIMUM_WAIT_TIME + TIMER;
X = ADDR(TCP);
Y = TCP;
                                                                                                                                                                                                                                                                            FINI: P->PCB.TIMER_CHAIN_LINK = Y;
X->LIST_ELEMENT.LINK = P;
P->PCB.BLOCKED_SEMAPHORE_PTR = S;
                                                                                                                                                                                                                         Y = Y->PCB.TIMER_CHAIN_LINK;
LINK_TC: PROCEDURE(P,S);
DECLARE (P,S,X,Y) POINTER;
                                                                                                                                                                                                                                                                                                                                                                END LINK_TC;
                                                                                                                                                                                                                                                   GOTO LOOP;
```

```
UNLINK_TC: PROCEDURE(X):

DECLARE (X,Y) POINTER;

IF TCP = X

THEN TCP = TCP->PCB.TIMER_CHAIN_LINK;

ELSE DO:

Y = TCP;

DG WHILE (Y->PCB.TIMER_CHAIN_LINK ~= X);

Y = Y->PCB.TIMER_CHAIN_LINK;

END;

Y->PCB.TIMER_CHAIN_LINK = X->PCB.TIMER_CHAIN_LINK;

END;

X->PCB.TIMER_CHAIN_LINK = NULL;

END;
```

Figure III-12. The procedure UNLINK\_TC.

The NUCLEUS provides safeguards to insure that the only primitives that may access event channels are SIGNAL, BLOCK, and CHECK, and that the only primitives that may access message queues are SEND, RECEIVE, and WAIT. Since the nature of those safeguards is irrelevant to the purpose of this work, we will not discuss them.

There is, however, one additional procedure which accesses the semaphores, and that is the real-time clock interrupt handler. If a process specifies a maximum-wait-time on a BLOCK or WAIT primitive and the process must be blocked, in addition to being placed on the queue of the semaphore, it is also linked onto the timer chain. The specified maximum-wait-time is added to the system time, i.e. TIMER, at the time of the call to obtain the time-out-time, the earlier time-out-times nearer the head of the list. Each time the real-time clock interrupt occurs, TIMER is incremented and all processes on the timer chain which have time-out-times less than TIMER will be timed-out i.e., they will be removed from the timer chain, removed from the queue of the semaphore upon which they are blocked and added to the set of ready processes. Status information will be returned indicating that the process timed-out. Note that in addition to removing a process from a semaphore's queue, the counter of the semaphore must be incremented to preserve the integrity of the system.

Figures III-2 through III-12 are descriptions in PL/I of the procedures of the NUCLEUS that implement the process interaction primitives. Due to the descriptive power of PL/I and the fact that these procedures have already been more or less described, no detailed descriptions will be given. It should be noted that procedures ENQUEUE and DEQUEUE are used

to link PCB's and message buffers onto priority/time queues, or to remove them from queues, respectively. Also, the procedures LINK\_TC and UNLINK\_TC are used to link PCB's onto the timer chain or to remove them from it.

The procedures SIGNAL, BLOCK, CHECK, SEND, WAIT, and RECEIVE are called by the monitor interrupt handler in response to processes' calls for NUCLEUS services. The procedures ENQUEUE, DEQUEUE, LINK\_TC and UNLINK\_TC are called by the service procedures listed above. The procedure RECINT is called at a 60 Hz rate in response to the real-time clock interrupt.

### Our Model of the NUCLEUS

In this section we present our model of the NUCLEUS. We model the procedures of the NUCLEUS described in the previous section as six actions, three of type SYNCHRONIZER and three of type COMMUNICATOR. These actions may be combined with other actions to form a multiprogramming system as defined in Chapter II.

At this point we emphasize that our model of the NUCLEUS is an abstraction. In fact, our description of the NUCLEUS in PL/I is also an abstraction. At present there is no method for determining whether an abstraction truly characterizes the thing it purports to abstract. Human insight must be used to give confidence in the abstraction. In this work we are primarily interested in proving that a given abstraction possesses particular operational characteristics and not in trying to prove the validity of the given abstraction. Thus, we simply state that we have confidence in our abstraction presented below and leave it at that.

Our abstraction of the SIGNAL primitive is in the following action:

SIGNAL = WHEN 
$$L_i = j$$
 DO  $L_i = k$ ;  
ec = ec + 1;  
stat = 0;

where i is the number of the process containing the action, j is the address of the action, k is the address of the next action to be executed, ec is an integer-valued data item in E, and stat is binary-valued data item in  $P_i$ . The SIGNAL action is in READY-SET( $\underline{a}$ ) whenever it is in POINTER-SET( $\underline{a}$ ) and thus may be executed immediately. The primary function of the SIGNAL primitive, namely incrementing the event channel specified, is accomplished by the assignment ec = ec + 1. ec corresponds to the counter of an event channel. Note that the release of any processes blocked on the event channel is not performed by the SIGNAL action. We see below that the release is embodied in the action predicate of the BLOCK action which resulted in a process being blocked. The data item stat is always set to zero by a SIGNAL action.

Our abstraction of the BLOCK primitive is the following action:

BLOCK= WHEN 
$$L_i = j\Lambda$$
 (ec > 0 V T > tot) DO  $L_i = k$ ; ec = IF ec > 0 THEN ec - 1 ELSE ec; stat = IF ec > 0 THEN 0 ELSE 1;

where i is the number of the process containing the action, j is the address of the action, k is the address of the next action to be executed, ec is an integer-valued data item in E, tot is the time-out-time of the process and stat is a binary-valued data item in  $P_i$ . The value of tot is the sum of the system time at which the action is added to POINTER-SET(a)

and the maximum-wait-time specified by the calling process. A BLOCK action can be moved from  $BLOCKED-SET(\underline{a})$  to  $READY-SET(\underline{a})$  for two reasons: the event blocked upon occurs (ec > 0) or the process times-out (T > tot). If the event occurs, ec is decremented and stat is given the value 0. If the process times-out, ec is unchanged and stat is given the value 1. Due to the implementation of the NUCLEUS, both cases cannot occur simultaneously.

Our abstraction of the CHECK primitive is the following action:

CHECK = WHEN 
$$L_i = j$$
 DO

L<sub>i</sub> = k; ec = IF ec>0 THEN ec-1 ELSE ec; stat = IF ec>0 THEN 0 ELSE 1;

where i is the number of the process containing the action, j is the address of the action, k is the address of the next action to be executed, ec is an integer-valued data item in E, and stat is a binary-valued data item in  $P_i$ . The similarity between the CHECK and BLOCK actions may be misleading. The use of the condition ec>0 in the action function of a BLOCK action merely serves to determine whether an event occurred or the process timed out. Also, a BLOCK action may remain a member of BLOCKED-SET( $\underline{a}$ ) for a long time. The CHECK operation never enters BLOCKED-SET( $\underline{a}$ ).

Our abstraction of the SEND primitive is the following action:

where i is the number of the process containing the action, j is the address of the action, k is the address of the next action to be executed, mq is a composite data item in M consisting of mq.c, and integer-valued data item, and mq.q, a data item of type queue, pri is an integer valued data item from  $P_i$ , enqueue is a function which returns the queue formed by adding the data item specified by its first argument to the queue specified by its second argument according to the priority given by its third argument, and stat is a binary-valued data item in  $P_i$ . Note that mq corresponds to a message queue, mq.c to the counter and mq.q to the queue. The enqueue function puts mess on mq.q according to the algorithm described in the procedure ENQUEUE, i.e. mq.q is ordered on the basis of the priority of the sending process, and, among groups with equal priorities, on the basis of the arrival time in the queue. The data item mq.q then is a queue in which messages may be buffered.

Our abstraction of the RECEIVE primitive is the following action:

RECEIVE = WHEN L<sub>i</sub> = j DO

L<sub>i</sub> = k;

mq.c = IF mq.c>0 THEN mq.c-1 ELSE mq.c;

mess = IF mq.c>0 THEN head(mq.q) ELSE 0;

mq.q = IF mq.c>0 THEN tail(mq.q) ELSE mq.q;

stat = IF mq.c>0 THEN () ELSE 1;

where i is the number of the process containing the action, j is the address of the action, k is the address of the next action to be executed, mq is a composite data item in M consisting of mq.c, an integer-valued data item, and mq.q, a data item of type queue, mess is a 64-bit data item in

 $P_i$ , head is a function which returns the first element of the queue specified as its argument, tail is a function which returns the queue specified as its argument minus its head, and stat is a binary-valued data item in  $P_i$ . Note that mq corresponds to a message queue, mq.c to its counter and mq.q to its queue.

Our abstraction of the WAIT primitive is the following action:

WAIT = WHEN 
$$L_i = j\Lambda(mq.c > 0 \text{ V T > tot}) D0$$
  
 $L_i = k;$   
 $mq.c = IF mq.c > 0 \text{ THEN } mq.c - 1 \text{ ELSE } mq.c;$   
 $mq.q = IF mq.c > 0 \text{ THEN } tail(mq.q) \text{ ELSE } mq.q;$   
 $mess = IF mq.c > 0 \text{ THEN } head (mq.q) \text{ ELSE } 0;$   
 $stat = IF mq.c > 0 \text{ THEN } 0 \text{ ELSE } 1;$ 

where i is the number of the process containing the action, j is the address of the action, k is the address of the next action, mq is a composite data item in M consisting of mq.c, an integer-valued data item, and mq.q, a data item of type queue, tot is the time-out-time of the process, mess is a 64-bit data item in  $P_i$ , head is a function that returns the first element of the queue specified as its argument, tail is a function which returns the queue specified as its argument minus its head, and stat is a binary-valued data item in  $P_i$ . The value of tot is the sum of the system time at which the action is added to  $POINTER-SET(\underline{a})$  and the maximum-wait-time specified by the calling process.

Note that the procedure RTCINT is not explicitly included in our model of the NUCLEUS. Its function is split between the action predicates of the BLOCK and WAIT actions.

Our model of the NUCLEUS specifically includes the queueing discipline used for messages. However, it omits the queueing discipline for processes blocked upon semaphores and for processes waiting on the Ready Process Queue. We may state these queueing disciplines in the form of a scheduler which is true only for computations which embody the queueing disciplines of the NUCLEUS. Since the proofs of correctness in Chapter V are independent of the scheduler used, we omit precise definition of the scheduler, and simply assert its existence.

#### IV. CORRECTNESS CONCEPTS

#### Introduction

In this chapter we discuss several correctness criteria, i.e. statements about a multiprogramming system which, if true, imply that the system operates as desired. In Chapter V we will use the correctness criteria described in this chapter to prove the correctness of the NUCLEUS described in Chapter III. It must be emphasized that correctness is not an absolute, and may only be asserted with respect to a particular set of criteria. Selection of adequate criteria is made by applying human intuition to some sort of set of specifications that describe the way the system should operate. Proof that correctness criteria are satisfied then implies that the system does meet its specifications.

We are interested, in this work, not in proving the correctness of a particular set of processes with respect to some criteria, but in proving that a particular set of synchronization and interprocess communication primitives will allow certain types of system behavior while disallowing certain other types of behavior. We would like to be able to make informal statements about the NUCLEUS of our example system of the form "If the primitives of the NUCLEUS are used in this manner, then the behavior of a system of processes will have these characteristics." To achieve this goal, we use partially interpreted program schemas (see Elspas, et.al. [1972]). A program schema is an abstract program which represents only the form or structure of a computational process without regard to the particular

functions, predicates, or types of data items involved. An interpretation assigns particular functions, predicates, and types of data items to the schema. A partial interpretation leaves certain parts of the schema uninterpreted. In our schemas, the interpreted part will usually be actions used for synchronization or interprocess communication, and the uninterpreted part will be arbitrary actions not involved with synchronization or interprocess communication.

Our proofs in Chapter V will then consist of proofs of statements about the characteristics of schemas. The remainder of this chapter presents the schemas to be used in Chapter V and the statements about their behavior which will be proved.

### Basic Assumptions

Before presenting the criteria that must be proved, we must make several assumptions about the operation of the system. The processes of the system are assumed to contain no errors. They are assumed to use the process interaction primitives only in the manner defined by our schemas. If the average processing load is greater than the capabilities of the machine for significant lengths of time, the system may fail. The priority scheduling policy of the NUCLEUS was selected to allow processes with fast response time requirements to have high priorities, i.e., to be run before other ready processes with lower priorities. Processes with lower response time requirements may be assigned lower priorities and thus be run when no process of higher priority is ready. However, when the system is overloaded some of the lower priority processes may never be run due to the unavailability of CPU cycles, thus degrading system performance significantly

or possibly causing complete system failure. Since we are interested in making statements about the logical structure of the system and not about its response under overload conditions, we restrict the system to running only sets of processes which do not overload the system under the given operating conditions. Due to the imprecise nature of overload, we can not make a formal definitive statement of thes restriction. However, we may make the following informal statement: "Every process that is logically able to proceed, will eventually proceed."

We make another assumption to facilitate proofs of existence. We assume that every process in a multiprogramming system is <u>cyclic</u>. Also we assume that the system of processes runs "forever." This is not strictly true since the operator may halt the system at his desire. However, until the instant the system is halted, it runs as if it were going to run forever. After the system is halted, the state of the system is irrelevant. We assume that the implementation of the system matches the abstraction of it and thus no errors caused by "bugs" in the NUCLEUS can occur. Our confidence in the truth of this assumption is based on the performance of a series of tests both on the individual modules of the NUCLEUS and on the NUCLEUS as a whole.

# Sequentiality

The first correctness criteria we state is independent of the form of a process. One of the advantages of a multiprogramming operating system is that it permits a complex, non-deterministic system to be written as a set of deterministic sequential programs. We use the term non-deterministic in the sense that, for the system as a whole, we cannot know a priori the

order in which the instructions will be executed. However, the individual processes of the system are deterministic in the sense that their instructions are executed sequentially. We formalize this notion by stating the following definition:

A set of actions in a multiprogramming system is <u>sequential</u> if for every finite computation  $\underline{a}$ , at most one action from this set is in READY-SET( $\underline{a}$ ).

We are now ready to state our first criterion of correctness:

<u>Criterion 1</u>. Each process in a multiprogramming system is sequential.

This will be proven in Chapter V.

## Producer-Consumer Message Passing

An important type of process interaction is the production and comsumption of tasks as in Dijkstra [1968b]. One process generates tasks which are communicated to another process which performs the task. In general, there is one consumer process and one or more producer processes. Both producers and consumers are cyclic. Each producer builds a description of a task and sends it to a message queue, then builds another, etc. The consumer waits until a task arrives in the message queue. It then receives the task and processes it. After processing, it waits for another task.

Figure IV-1 shows a program schema for the interaction described above. Action 1 of the consumer is a WAIT primitive. The message queue involved is named "mailbox," which is somewhat indicative of its use. Action 2 is the consumption of the message (or processing of the task it represents).

```
DATA:
   (stat<sub>1</sub>, stat<sub>2</sub>..., stat<sub>k</sub>) BIT;
   (L_1, L_2, ... L_k) PROGRAM-COUNTER INITIAL(1,3,...,2k-1);
   mailbox STRUCTURE(c INTEGER INITIAL(0),q QUEUE INITIAL(Λ));
   (mess, mess, ) MESSAGE;
ACTIONS:
   CONSUMER
      (1) WHEN (L_1=1) \land (mailbox>0) DO L_1=2;
                                                                /WAIT(mailbox, mess<sub>1</sub>)
                                           mailbox.c=mailbox.c-1;
                                           mailbox.q=tail(mailbox.q);
                                           mess<sub>1</sub>=head(mailbox.q);
                                           stat,=0;
      (2) WHEN L1=2 DO L1=1;
                           consume mess;
PRODUCER-i (i=1,2,...,k-1)
      (2i+1) WHEN L<sub>i+1</sub>=1 DO L<sub>i+1</sub>≈2i+2;
                                produce mess<sub>i+1</sub>;
      (2i+2) WHEN L<sub>i+1</sub>=2 DO L<sub>i+1</sub>≈2i+1;
                                                             /SEND(mailbox, mess;+1)
                                mailbox.c=mailbox.c+l;
                               mailbox.q=enqueue(mess_{i+1}, mailbox.q, pri_{i+1});
                               stat;+1=0;
```

Figure IV-1. Program Schema for Producer/Consumer Message Passing.

Although we show only one action for the consumption, extension to multiple actions is trivial. The same is true for action 2i+1 of PRODUCER-i, where generation of the taks takes place. Action 2i+2 is a SEND primitive.

Our criterion for correct operation of this schema is that the consumer cannot consume a message until it has been produced. We state this formally as:

<u>Criterion 2</u>. For every finite computation of the multiprogramming system described in Figure IV-1, the number of executions of actions 2i+1 for all i is greater than or equal to the number of executions of action 2.

### Time-out Processing

The time-out feature of the BLOCK and WAIT primitives is useful in cases where a process expects to be awakened within a finite period of time. E.g., hardware malfunction in an I/O device could cause a process waiting for an interrupt to be blocked forever if the time-out feature is not used and the interrupt is not generated. (Interrupts are mapped into SIGNAL operations.)

Figure IV-2 shows a program schema for a simulated I/O device with random failures and its driver. Action 1 is a WAIT on the event channel "command." Action 2 examines the random valued bit named "fail." If the device fails, no SIGNAL is done and the process simply waits for another command. Action 4 SIGNAL's the DRIVER that the I/O operation is complete. Action 5 SIGNAL"s the DEVICE to begin I/O. Action 6 is a BLOCK which causes the DRIVER to wait for the SIGNAL caused by the I/O interrupt or the specified time interval to pass. Action 7 determines whether the

```
DATA:
   (L<sub>1</sub>,L<sub>2</sub>) PROGRAM-COUNTER INITIAL(1,5);
   (command, interrupt) INTEGER INITIAL(0,0);
   (stat1, stat2, fail) BIT:
   tot, INTEGER;
ACTIONS:
   DEVICE
     (1) WHEN (L_1=1)_{\Lambda} (command>0) D0 L_1=2;
                                                                /WAIT(command)
                                       command=command-1;
                                       stat<sub>1</sub>=0;
     (2) WHEN L_1 \approx 2 DO L_1 = IF fail THEN 3 ELSE 4;
     (3) WHEN L_1=3 DO L_1=1;
                                                                 /I/O failure
     (4) WHEN L_1=4 DO L_1=1; /SIGNAL(interrupt) - I/O complete
                       interrupt=interrupt+1;
                      stat<sub>1</sub>=0;
DRIVER
     (5) WHEN L_2=5 DO L_2=6; /SIGNAL(command) - begin I/0
                       command=command+1;
                       stat_=0;
     (6) WHEN (L_2=6)\Lambda(interrupt>0\Lambda T>tot_2) DO L_2=7; /BLOCK(interrupt)
               interrupt=IF interrupt>O THEN interrupt-1 ELSE interrupt;
               stat2=IF interrupt>0 THEN 0 ELSE 1;
     (7) WHEN L_2=7 DO L_2=IF stat_2=0 THEN 8 ELSE 9;
     (8) WHEN L_2=8 DO L_2=5; do interrupt processing; /I/O termination
     (9) WHEN L_2=9 DO L_2=5; do no-interrupt processing; /time-out
```

Figure IV-2. Program Schema for I/O Device and Driver.

interrupt or time-out occurred and branches to action 8 to perform normal I/O termination processing or to action 9 to perform failure processing.

Our notion of correctness for the time-out processing schema is that every time action 5 is executed, either action 8 or 9 will be executed. We formalize this as:

<u>Criterion 3.</u> For every computation  $\underline{a}$  of the multiprogramming system shown in Figure IV-2, the following are true:

- a. If  $\underline{a}_i$  is action 3 for any i, then  $\underline{a}_{i+3}$  will be action 9.
- b. If  $\underline{a}_i$  is action 4 for any i, then  $\underline{a}_{i+3}$  will be action 8.

Since the operation of the BLOCK and WAIT operations is so similar, we prove only that the time-out feature works correctly for the BLOCK primitive.

## No-block Synchronization

Some types of events in multiprogramming systems occur infrequently and do not require a fast response time. To avoid the overhead of having to wait for an event of this type, code may be added to an existing, frequently run process to check for the event and perform the required processing if the event has occurred. Since this process cannot allow itself to be blocked for a potentially long time, it uses no-block synchronization, i.e. it uses the CHECK primitive rather than the BLOCK primitive. It may CHECK for the occurrence of several events.

Figure IV-3 shows a program schema that demonstrates the situation above. PROCESS-1 signals the event channel "rare" on an infrequent basis. Action 1 is an action which provides the long interval between event occurrences. Action 2 is a SIGNAL operation on the event channel "rare."

```
DATA:
   (L1,L2) PROGRAM-COUNTER INITIAL(1,3);
   (stat1, stat2) BIT;
   rare INTEGER INITIAL(0);
ACTIONS:
   PROCESS-1
      (1) WHEN L_1 = 1 DO L_1 = 2;
                         do something for a long time;
      (2) WHEN L_1 = 2 DO L_1 = 1;
                                                                      /SIGNAL(rare)
                          rare-rare+1;
                         stat<sub>1</sub>=0;
   PROCESS-2
      (3) WHEN L<sub>2</sub>=3 DO L<sub>2</sub>=4;
                                                                      /CHECK(rare)
                         rare=IF rare>0 THEN rare-1 ELSE rare;
                         stat2=IF rare>0 THEN 0 ELSE 1;
      (4) WHEN L_2=4 DO L_2=IF stat_2=0 THEN 5 ELSE 6;
      (5) WHEN L2=5 DO L2=6;
                         processing for rare event;
```

Figure IV-3. Program Schema for No-block Synchronization.

remainder of processing;

(6) WHEN L2=6 DO L2=1;

PROCESS-2 periodically checks "rare" to determine if the event has occurred. If it has, the required processing is done. Action 3 is a CHECK operation on the event channel "rare." Action 4 examines "stat<sub>2</sub>" to determine whether or not the event has occurred. Action 5 is executed if the event occurred. Action 6 is the processing done during the remainder of PROCESS-2. We make the assumption that the time between executions of action 1 is much greater than the time required for one cycle of the actions in PROCESS-2.

In order to prove the correctness of the multiprogramming system in Figure IV-3, we must prove that every time action 2 is executed, action 5 will be executed exactly one time. We state this formally as:

<u>Criterion 4</u>. For every computation <u>a</u> of the multiprogramming system in Figure IV-3, each occurrance of action 2 is followed by exactly one occurrance of action 5.

### Mutual Exclusion

The term mutual exclusion, used in connection with multiprogramming operating systems, means prohibiting any one of a set of cooperating processes from accessing a (possibly composite) data item while any other process in the set is also accessing it. Otherwise, the shared data structure would likely be corrupted. A common solution to the mutual exclusion problem is to include the instructions which act on the shared data in <a href="mailto:critical sections">critical sections</a> (see Dijkstra [1968a]), and allow only one process at a time to execute its critical section. Critical sections may be implemented by using the SIGNAL and BLOCK operations provided by the NUCLEUS. Figure IV-4 shows the form of one of a set of cooperating processes using critical sections. Note that we assume that each of the processes is

```
DATA:
   (L_1, L_2, ..., L_k) PROGRAM-COUNTER INITIAL(1,5,...,4k-3);
   mutex INTEGER INITIAL(1);
   (stat<sub>1</sub>,stat<sub>2</sub>,...,stat<sub>k</sub>) BIT;
ACTIONS:
   PROCESS-i (i=1,2,...,k)
       (4i-3) WHEN L<sub>i</sub>=4i-3 DO L<sub>i</sub>=4i-2;
                                  operate on non-shared data;
       (4i-2) WHEN (L_i=4i-2)\land (mutex>0) DO L_i=4i-1;
                                                                        /WAIT(mutex)
                                                  mutex=mutex-1;
                                                 stat;=0;
       (4i-1) WHEN L_i=4i-1 DO L_i=4i;
                                  operate on shared data;
       (4i)
             WHEN L; =4i DO L; =4i-3;
                                                                        /SIGNAL(mutex)
                                mutex=mutex+1;
                                stat; ≈0;
```

Figure IV-4. Program Schema for Mutual Exlcusion.

cyclic. Action 4i-3 includes all of the processing outside the critical section. Action 4i-2 is a BLOCK(mutex) operation. Action 4i-1 consists of the actions on shared data. Action 4i is a SIGNAL(mutex) operation. We assume that there are k processes and that the operations on both the shared and non-shared data may be different for each process. Although the operations on data (except "mutex") are shown as one action, it is trivial to extend the schema to any number of such actions.

Our correctness criterion for the schema in Figure IV-4 is the mutual exclusion of the critical sections. We will prove that at most one process at a time is in its critical section. We state this formally as:

<u>Criterion 5</u>. For every computation of the multiprogramming system described in Figure IV-4, if any action of the computation is action 4i-2 for any i, then it is immediately followed by the sequence of actions:

b, 4i-1, c, 4i

where  $\underline{b}$  and  $\underline{c}$  are finite, possibly null, sequences of actions not containing action 4j-2, 4j-1, or 4j, for all  $j \neq i$ , or any action from process i.

#### V. PROOFS OF CORRECTNESS

#### Introduction

In this chapter we prove that our correctness criteria for the program schemas presented in Chapter IV are satisfied. We present each criterion as a theorem and then prove the theorem.

### Proof of Sequentiality

THEOREM 1. Every process in a multiprogramming system is sequential.

PROOF: From the definition of sequential, each process must have at most one action in READY-SET( $\underline{a}$ ), for every computation  $\underline{a}$ . Assume that this is not true, i.e. some process has two actions, f and g, in READY-SET( $\underline{a}$ ) for some computation  $\underline{a}$ . From the definition of READY-SET( $\underline{a}$ ), the actions f and g are also in POINTER-SET( $\underline{a}$ ). Since f is in POINTER-SET( $\underline{a}$ ), then

 $L_{PROGRAM-COUNTER(f)} = ADDRESS(f).$ 

Since g is in POINTER-SET( $\underline{a}$ ), then

 $L_{PROGRAM-COUNTER(g)} = ADDRESS(g).$ 

Since f and g are in the same process,

PROGRAM-COUNTER(f) = PROGRAM-COUNTER(g).

Hence.

ADDRESS(f) = ADDRESS(g).

From condition 4 of the definition of multiprogramming system, we find that f = g.

This is in contradiction to the assumption that two actions of the process are in READY-SET( $\underline{a}$ ). Therefore, the theorem is proven.

## Proof of Producer-Consumer Message Passing

THEOREM 2. For every finite computation of the multiprogramming system described in Figure IV-1, the number of executions of action i-1, for all i, is greater than or equal to the number of executions of action 2.

PROOF: For the finite timing  $\underline{a}$ , let  $N(\underline{a},i)$  denote the number of occurrences of action i in computation  $\underline{a}$ . The proof now proceeds in 4 steps:

- 1. First we prove that mailbox.c>0. Its value is initially 0, which satisfies the stated relation. Only k actions of the system may change the value of mailbox.c: action 1 and actions 2i+2, for i=1,2,...,k-1. The latter may only increase the value of mailbox.c, so that if the relation holds before executing action 2i+2 for some i, it must also hold after executing it. The former decrements the value, but only when mailbox.c>0. Thus if the relation mailbox.c>0 holds before executing action 1, then mailbox.c>0 holds after executing it. If mailbox.c=0, action 1 cannot be executed. Thus, by inducation we have proven that mailbox.c>0.
  - 2. We now prove that

$$N(\underline{a},2i+2) - (N\underline{a},1) = mailbox.c(STATE(\underline{a})).$$

Observe that each execution of action 2i+2, for any i, increments mailbox.c and each execution of action 1 decrements mailbox.c. For the computation  $\underline{a}$ , mailbox.c will be incremented  $N(\underline{a},2i+2)$  times and decremented  $N(\underline{a},1)$  times. Since the initial value of mailbox.c is 0, its value after the computation  $\underline{a}$  is

mailbox.c(STATE(
$$\underline{a}$$
)) = N( $\underline{a}$ ,2i+2)(+1)+N( $\underline{a}$ ,1)(-1)  
= N( $\underline{a}$ ,2i+2) - N( $\underline{a}$ ,1)

#### 3. Now we prove that

$$N(\underline{a}, i-1)-N(\underline{a}, 2)>N(\underline{a}, i-2)-N(\underline{a}, 1)$$
.

Since the initial value of  $L_1$  is 1 and since the Consumer process is cyclic, the quantity  $N(a,1)-N(\underline{a},2)$  alternates in value between 0 and 1 as actions from the Consumer process are executed. Thus,

$$N(\underline{a},1)>N(\underline{a},2)$$
.

Similarly, it can be shown that  $N(\underline{a},2i+1) \ge N(\underline{a},2i+2)$ . Consider the quantity  $t = N(\underline{a},2i+2) - N(\underline{a},1)$ .

Since  $N(\underline{a},1)>N(\underline{a},2)$ ,

$$t < N(\underline{a}, 2i + 2) - N(\underline{a}, 2)$$
.

Also, since  $N(\underline{a},2i+1) > N(\underline{a},2i+2)$ ,

$$t < N(a, 2i+1) - N(a, 2)$$
.

Therefore,

$$N(\underline{a},2i+1) - N(\underline{a},2) > N(\underline{a},2i+2) - N(\underline{a},1)$$
.

4. Now we combine the results of steps 1 through 3 to arrive at the desired result:

$$N(a,2i+1) - N(a,2) > 0$$

or

$$N(\underline{a},2i+1) \geq N(\underline{a},2)$$
.

## Proof of Time-out Processing

THEOREM 3. For every computation  $\underline{a}$  of the multiprogramming system shown in Figure IV-2, the following is true:

- a. If  $\underline{a}_i$  is action 3 for any i, then  $\underline{a}_{i+3}$  is action 9.
- b. If  $\underline{a}_i$  is action 4 for any i, then  $\underline{a}_{i+3}$  is action 8.

PROOF: We will prove that the system given is cyclic in the sense that it proceeds from a "home" state where  $L_1=1$ ,  $L_2=5$ , command=0 and interrupt=0 through a sequence of intermediate states back to the home state. Then we will prove that for one cycle either condition a or condition b above is true. By symmetry of cycles, we then assert the truth of this theorem. Before continuing, we make one assumption that will be true in practice. We require that tot, be greater than the latest time at which action 4 may occur in one cycle of the system. This corresponds to specifying a maximum wait time on a BLOCK operation greater than the latest possible time for a service interrupt to be generated. Now we will prove the cyclic nature of the system. Initially,  $L_1$ =1 and  $L_2$ =5. Action 1 is blocked since command=0. Thus action 5 is executed. Now action 6 is blocked since interrupt=0 and by our assumption  $T \le tot_2$ . So action 1 must be executed, reducing command to 0. Then action 2 is executed, followed by either action 3 or 4. At this point  $L_1=1$  and  $L_2=6$  and action 1 is blocked since command=0. Action 6 must be the next action to be executed, either because action 4 was the previous action or becaust T>tot2. Then action 7 is executed, followed by either action 8 or 9. At this point the system has returned to its home state, namely  $L_1=1$ ,  $L_2=5$ , command=0 and interrupt=0, and the procedure described above is repeated cyclicly. Thus we see that one cycle of the system consists of the action sequence

5, 1, 2, x, 6, 7, y

where x is action 3 or 4 and y is action 8 or 9. If x is action 4, then interrupt will be assigned the value 1, and action 6 will assign interrupt the value 0 and stat<sub>2</sub> the value 0. The execution of action 7 will cause

action 8 to be its successor. Thus y is action 8. If, however, action 3 is executed, action 6 will be executed only after T>tot<sub>2</sub>. Since interrupt=0, stat<sub>2</sub> is assigned the value 1. Then action 9 will be the successor of action 7. Thus y is action 9. The cycles of the system are thus seen to be of two forms:

5, 1, 2, 4, 6, 7, 8

or

5, 1, 2, 3, 6, 7, 9.

Since all cycles must be of one of these forms, the theorem is proved.

## Proof of No-block Synchronization

THEOREM 4. For every computation of the multiprogramming system in Figure IV-3, each occurrance of action 2 is followed by exactly one occurrence of action 5.

PROOF: Observe that PROCESS-2 is cyclic. Until the occurrence of the rare event, i.e. the execution of action 2, PROCESS-2 cycles through action 3, 4, 6, 3, .... On the first occurrence of action 3 after an execution of action 2, rare is decremented (to 0) and stat<sub>2</sub> is set to 0. Then actions 4, 5, and 6 are executed in order. On the second and subsequent executions of action 3 after action 2 but before another execution of action 2, rare will be 0 and thus stat<sub>2</sub> will be set to 1 and action 5 will not be included in the cycle. Thus we see that execution of action 2 causes exactly one execution of action 5.

## Proof of Mutual Exclusion

THEOREM 5. For every computation of the multiprogramming system described in Figure IV-4, if any action of the computation is action 4i-2 for any i, then it is immediately followed by the sequence:

b,4i-1, c, 4i

where  $\underline{b}$  and  $\underline{c}$  are finite, possibly null, sequences of actions not containing action 4j-2, 4j-1, or 4j for all  $j \neq i$ , or any action from process i.

PROOF: Assume there are k processes in the multiprogramming system. We say that the system is non-critical if  $((L_i=1)V(L_i=2))$  mutex=1 for i=1,2,...,k. Otherwise, we say that the system is critical. We will prove by mathematical induction that the system is initially non-critical and progresses cyclicly from non-critical to critical to non-critical, etc., producing only computations of the form stated in this theorem. It is obvious that the system is non-critical initially since  $L_i=1$ , for i=1,2,...,k, and mutex=1. We now present our induction proof. Assume that the system is non-critical. Then, in general,  $L_i=1$  for some values of i and  $L_i=2$  for all other values of i. Since mutex=1, no action can be blocked with its L<sub>i</sub>=2. There is a finite limit on the number of actions that may be executed before action 4i-2 is executed for some i. To prove this, note that there are k processes in the system and hence k program counters. Then the number of program counters which equal 1 is less than or equal k. In the worst case, all of the actions pointed to by this set of program counters will be executed before some action 4i-w is executed. Execution of action 4i-3 for any i will not cause the system to change from non-critical to

critical, since  $L_i$  changes from 1 to 2 and mutex cannot be changed by action 4i-3. Whenever action 4i-2 for some i is executed, the system changes from non-critical to critical, since both L; changes to 3 and mutex changes to 0. Let a represent the sequence of actions from the beginning of the computation up to the action 4i-2. Then all actions whose program counters equal 1 are in READY-SET(a,4i-2) and all actions whose program counters equal 2 are in BLOCKED-SET(a,4i-w). No program counter can equal 4 and none but the program counter of process i can equal 3. Hence, the next actions that may be executed are actions 4j-3 for some j, and action 4i-1. The number of actions that may be executed before executing action 4i-1 is bounded. Call the sequence of these actions b. The execution of action 4i-1 does not change the system back to non-critical since it does not change mutex. After ex-cution of action 4i-1, READY-SET(a,4i-2,b,4i-1) consists of only those actions whose program counter equals 1 and action 4i. Again, the number of actions that can be executed before executing action 4i is bounded. Call the sequence of these actions  $\underline{c}$ . Note that execution of action 4i changes  $L_i$  to 1 and mutex to 1, thus changing the system from critical to non-critical. Therefore, for the multiprogramming system shown in Figure IV-4, if the system is non-critical, it will become critical within a finite number of actions, and then will again become non-critical within a finite number of actions.

In showing the cyclic nature of the system, we have proved Theorem 5, namely that whenever action 4i-2 for some i is executed, it is followed by the sequence  $\underline{b}$ ,4i-1, $\underline{c}$ ,4i where  $\underline{b}$  and  $\underline{c}$  are finite, possibly null, sequences of actions not containing any actions from process i or any actions 4j-2,4j-1, or 4j for all  $j \neq i$ .

AUBURN UNIV ALA ENGINEERING EXPERIMENT STATION REAL-TIME PHASED ARRAY RADAR STUDIES.(U) 1976 E R GRAF, H T NAGLE AD-AU32 121 F/G 17/9 DAAH01-71-C-1303 NL UNCLASSIFIED 4 of 5 ABBRIZE

### VI. CONCLUSION

## Conc usions

In this work, we have presented a model useful for describing systems of computer programs which are executed in a parallel manner. We have used this model to prove the correctness of an example system with respect to certain criteria. The criteria were expressed in terms of partially interpreted program schemas so that the proofs given are applicable to a class of multiprogramming systems. Obviously, by giving proofs of additional criteria, we could have broadened the class of systems whose correctness has been proven.

The approach taken in our proof procedure allows expression of correctness criteria in terms of the temporal ordering of events rather than in terms of the state of the system data base. The former is a more natural way of expressing statements about the operation of a system. However, much intellectual effort is required to produce a proof. In methods based on the state of the system data base, mechanical algorithms exist for proving the correctness criteria.

# Suggestions for Future Research

Some areas where additional research would be useful include:

1. Inclusion of scheduling constraints in proofs. The proofs given in Chapter 5 contain no reference to the scheduling policies of the multiprogramming system. Development of a more convenient means of expression of the concept of schedulers would allow "stronger" correctness criteria to be expressed and proved.

- 2. In the proofs of Chapter V it was assumed that the system was not overloaded. Techniques need to be developed that allow proofs about the behavior of a multiprogramming system that is overloaded or nearly so.
- 3. It was assumed that the systems of processes used in the proofs were non-terminating. In practice, however, systems do terminate and, more importantly, parts of the system terminate while other parts continue to run. It would be useful to find some formalism of termination that would make it practical to make proofs about the behavior of a system during the various types of termination.
- 4. The proofs procedure could possibly be made easier by the development of concepts and theory that would make the expression of correctness criteria simpler.

### BIBLIOGRAPHY

- Ashcroft and Manna [1971]
  Ashcroft, E. and Z. Manna, "Formalization of Properties of Parallel Programs," Machine Intelligence 6, ed. Meltzer and Michie, University of Edinburgh Press, Edinburgh, 1971.
- Dennis and Van Horn [1966]

  Dennis, J. B. and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," Communications of the ACM, v. 9, n. 3,
  March 1966, pp. 143-155.
- Dijkstra [1968a]
  Dijkstra, E. W., "Co-operating Sequential Processes," Programming
  Languages, NATO Advanced Study Institute, ed. F. Genuys, Academic
  Press, New York, 1968, pp. 43-112.
- Dijkstra [1968b]
  Dijkstra, E. W., "The Structure of THE Multiprogramming System,"
  Communications of the ACM, v. 11, n. 5, May 1968, pp. 341-346.
- Elspas et al [1972]
  Elspas, B., K. N. Levitt, R. J. Waldinger and A. Waksman, "An Assessment of Techniques for Proving Program Correctness,"
  Computing Surveys, v. 4, n. 2, June 1972, pp. 97-147.
- Floyd [1967]
  Floyd, R. W., "Assigning Meanings to Programs," Mathematical
  Aspects of Computer Science, v. 19, ed. J. T. Schwartz, Amer.
  Math. Soc., Providence, R. I., 1967, pp. 19-32.
- Gilbert and Chandler [1972]
  Gilbert, P. and W. J. Chandler, "Interference Between Communicating Parallel Processes," Communication of the ACM, v. 15, n. 6, June 1972, pp. 427-437.
- Hawkins [1974]
  Hawkins, J. E., "Nucleus for an Experimental Array Radar Operating System," Technical Report, Engineering Experiment Station, Auburn University, Auburn, Alabama, September 1974.
- Kahn [1972]
  Kahn, G., "An Approach to Systems Correctness," Operating Systems Review, v. 6, n. 1, 2, June 1972, pp. 81-94.

Lauer [1973]
Lauer, H., "Correctness in Operating Systems," Ph.D. Thesis,
Carnegie-Mellon University, Pittsburgh, Pennsylvania, September,
1972.

Levitt [1972]
Levitt, K. N., "The Application of Program-proving Techniques to the Verification of Synchronization Processes," <a href="Proceedings of AFIPS 41">Proceedings of AFIPS 41</a>, FJCC, 1972, pp. 33-47.

Lipton [1973]
Lipton, R. J., "On Synchronization Primitive Systems," Ph.D. Thesis Carnegie-Mellon University, Pittsburgh, Pennsylvania, June 1973.

Naur [1966]
Naur, P., "Proof of Algorithms by General Snapshots," <u>BIT</u>, v. 6, n. 4, 1966, pp. 310-336.

Organick [1972]
Organick, E. I., The MULTICS System: An Examination of its Structure,
M.I.T. Press, Cambridge, Massachusetts, 1972.

## PART THREE

# MODULE PERFORMANCE

**MEASUREMENT** 

for

U.S. Army Missile Command Redstone Arsenal, Alabama

by

Engineering Experiment Station Auburn University Auburn, Alabama

Prepared by: Donald E. Wright Joe E. Hawkins

Reviewed by: H. Troy Nagle, Jr.

# TABLE OF CONTENTS

I.	Introduc	tion	1
II.	Reductio	n Program Description • • • • • • • • • • • • • • • • • • •	2
III.	ENQUEUE	Module Performance · · · · · · · · · · · · · · · · · · ·	11
	Attachme	nts	
	1.	Example Flowchart	13
	2.	Transformed Flowchart	14
	3.	Control Record · · · · · · · · · · · · · · · · · · ·	15
	4.	Data Record	16
	5.	Equations	17
	6.	Program Listing · · · · · · · · · · · · · · · · · · ·	18
	7.	Sample Input/Output	24
	8.	Program Flowchart	28
	9.	Flowchart of ENQUEUE	34
	10.	Transformed ENQUEUE Flowchart	35
Refere	nce		36

## I. INTRODUCTION

This report describes a technique [1] for reducing a flowchart, with single entry and exit points, to a single block described by an average execution time, expected memory requirements, etc. Here we summarize in section 2 the technique and present a FORTRAN listing of the flowchart reduction program. Finally in section 3, the reduction program is applied to a typical module (ENQUEUE) of the RADACS operating system.

## II. REDUCTION PROGRAM DESCRIPTION

This program provides the media for analysis of the mean and variance of a flow chart with respect to time or channel utilization.

A flow chart may be represented as a set of nodes and links. A node may have any number of links entering it (called incident links) and any number of links leaving it (called excident links).

A <u>decision</u> is a node with exactly one incident link and more than one excident link, while a <u>junction</u> is a node with one or more incident links and exactly one excident link. The sum of the excident link probabilities of every node must equal one. Associated with each link is a probability and weights representing the mean and variance with respect to time or channel utilization. Each node will have an interger name. Each link will be named by the nodes which it spans. Thus, if a link goes from node I to node J, it will be called link (I,J). If, on the other hand, it went from node J to node I, it would be called link (J,I). Similarly, the probability of a link and the weights associated with that link can be thought of as being designated by subscripts corresponding to the names of the nodes that the link spans.

The link above is an excident link from node I and an incident link to node J.

P(I,J) = probability of going to node J when at node I.

U(I,J) - mean value of time or channel utilization in link (I,J).

V(I,J) = variance of time or channel utilization in link (I,J).

Given the above conventions, any flow chart can be represented as a matrix in which the (I,J)th entry contains the information associated with the (I,J)th link. Each row represents the links that are excident to the node corresponding to that row, therefore the sum of the probabilities in each row must equal one. Each column represents the links that are incident to the node heading the column.

It is from this convention that the user of this program will develope the input file.

To further understand this procedure, consider the flow chart of Attachment 1. Associated with each symbol of the flow chart are two weights which represent its mean and variance with respect to time or channel unilization. The first step is to determine the number of nodes within the flow chart and assign them numerical names, secondly assign values to the weights associated with each link, and finally build the input file.

To accomplish step one, the start or inway symbol to the flow chart is one node and the stop or outway symbol another. The remaining nodes are the decision and junction nodes within the flow chart. Symbols B, D, and K of the flow chart each having two inputs and one output will determine the junction nodes, the decision symbols E, F, J and M the decision nodes. Therefore, the total number of nodes for the flow chart is nine. The

numerical names of the nodes of any flow chart will be 1, 2, 3, 4, ..., n, where n is the total number of nodes. The start or inway symbol of the flow chart <u>must</u> be given the numerical name 1. The stop or outway symbol <u>must</u> be given the numerical name 2. The remaining nodes will be given the numerical names 3, 4, 5, ..., n at the discretion of the user.

Attachment 2 will be used as an aid to understand how to complete the second step. It contains the flow chart with the nodes designated and will be used to specify the links and their associated weights. Link (1,3) is the only link spanning nodes 1 and 3, and is excident to node 1. Since it has bee previously stated that the sum of the excident link probabilities of every node must equal one and node 1 has one excident link, then the probability of link (1,3) must equal 1. The mean and variance of a link (I,J) is equal to the sum of the means and variances of the symbols after node I thru node J (if node J has mean and variance assignments). Therefore, the mean of link (1,3) is U(A) and the variance is V(A). The same procedure is used to determine the probabilities and weights of links (3,4) and (4,5). Considering link (5,7), node 5 has two excident links whose probabilities when summed must equal one. It is the responsibility of the user to make probability assignments to the excident links of a node to satisfy the above condition. To satisfy notation of the assignment of probabilites,  $P_{T}$  is equal to the probability of one of the excident links of node I and  $Q_I = 1 - P_I$  for the other. Thus,  $P_5$  will be the probability of link (5,7) and  $Q_5$  of link (5,6) or vice versa. The mean of link (5,7) is equal to U(H) + U(I) + U(J), while the variance is equal to V(H) + V(I) + V(J). This procedure is followed until all the links with their probabilites and associated weights, have been specified.

Finaly, each link with its probability and assigned weights will be the data for one record of the input file (see ATCH. 4). Note that the data in columns 002 thru 007 of Attachment 4 is the link name. Therefore, the number of records in the input file for the original flow chart is equal to the sum of the times links are specified plus a control card. The input file for the flow chart given, excluding the control card, is as follows:

Columns:	002-004	005-007	008-016	017-026	027-036
I	001	003	1.0	U(A)	V(A)
N	003	004	1.0	U(B)+U(C)	V(B)+V(C)
Р	004	005	1.0	U(D)+U(E)	V(D)+V(E)
U	005	006	P <sub>5</sub>	U(F)	V(F)
T	005	007	Q <sub>5</sub>	U(H)+U(I)+U(J)	V(H)+V(I)+V(J)
	006	002	P <sub>6</sub>	U(G)	V(G)
D	006	002	Q <sub>6</sub>	0	0
Α	007	004	P <sub>7</sub>	0	0
T	007	800	Q <sub>7</sub>	0	0
Α	008	009	1.0	U(K)+U(M)	V(K)+V(M)
	009	003	Q <sub>9</sub>	0	0
	009	800	Pg	U(L)	V(L)

The program takes the input file and builds three n x n matrices, where n is equal to the number of nodes within the flow chart. One matrix contains the probabilities of all the links, another the means, and the other the variances. Then each of these matrices is reduced using parallel

equations, loop equations and cross-term equations (see ATCH. 5) until there are three 2  $\times$  2 matrices. The three 2  $\times$  2 matrices are then transformed by the loop equations. The values of the weights of link (1,2) at this point in the reduction process, are the mean and variance of the flow chart, which is the objective of this program.

Looking at the input data derived from the flow chart given, the user notes that link (6,2) has two probability values and two values for each weight. The parallel equations will transform the multi-values of the probabilities and the associated weights of the link into single values. It is important to point out at this time that in reality, the three matrices mentioned above are three dimensional in the program instead of two. The third dimension takes care of links that have more than one value for the probability and the associated weights. For example, if the following data was part of the input file:

(1) 006 002 
$$P_6$$
 U(G) V(G)  
(2) 006 002  $Q_6$  0 0

The computer will store it in the appropriate matrices as follows:

$$P(6,2,1) = P_6$$
  $P(6,2,2) = Q_6$   
 $U(6,2,1) = U(G)$   $U(6,2,2) = 0$   
 $V(6,2,1) = V(G)$   $V(6,2,2) = 0$ 

The link or links that are input to the program the most times will determine the depth of the matrices, ie., the maximum size of the third dimension of the matrices. The depth is equal to the number of times the

link in question is input to the program. In the example, link (6,2) is the only link that is input to the program more times than the others. Thurs, the depth of the matrices from the flow chart is 2. This information must be entered on the control card (see ATCH. 3).

The first time the matrices are transformed by the parallel equations, the result is stored in 3 additional matrices to be used later to reevaluate the original flow chart. Next the matrices are transformed by the loop equations which eliminates the links that link a node to itself-for example, link (I,I). The third transformation using the cross-term equations will reduce the matrices to (N-1) X (N-1) matrices. The (N-1) X N-1) matrices are then taken throught the parallel equations, loop equations and the cross-term equations and reduced to (N-2) X (N-2) matrices. This process is continued until the matrices are reduced to 2 x 2 matrices. The 2 x 2 matrices are then taken throught the loop equations. The contents of link (1,2) at this time is the result of the evaluation of the flow chart.

If the user wants a listing of the transformation of the original matrices to a set of 2 X 2 matrices, a "1" must be punched in column 4 of the control card (see ATCH. 3). If the same is desired for the re-evaluations, a "1" must be punched in column 5 of the control card. In the transformations above where the n x n matrices are reduced to (N-1) X (N-1); (N-2) X (N-2); ...; 2 X 2 matrices, the information is printed after being transformed by the parallel equations when the criteria for the options is punched in the control card.

Attachment 7B contains a listing of the transformation of a set of 4 by

4 matrices to a set of 2 by 2 matrices. This is a transformation of an original flow chart as indicated by the header line at the top of the page. The values under the terms ROW and COLUMN of the header line represents the (I,J)th entries of the matrices. The value under the term PROBABILITY is the contents of the (I,J)th entry in the probability matrix. The values under the terms MEAN and VARIANCE, the contents of the (I,J)th entries of the mean and variance matrices. The value under the term MATRIX is the size of the matrices as they are being transformed. The (I,J)th entries of a set of N by N matrices contain the values for the probability and weights associated with the (I,J)th link. For example, on Attachment 7B the line with the value 1 under the term ROW, the value 3 under the term COLUMN, and the size 4 by 4 udner the term MATRIX contains the information associated with link (1,3) of this 4 by 4 matrix.

The result of the evaluation of the orginal flow chart mentioned above is shown at the bottom of the page.

Attachments 7C and 7D contain the transformations of re-evaluations of the original flow chart.

The user of this program can re-evaluate his flow chart, i.e., assign different probabilities and weights to one or more nodes in the original flow chart and evaluate the results if so desired. All that is required is for records to be punched reflecting the new values with the condition that column one of the <u>first record</u> of this group contain the value "1". These records are placed behind the input records of the original flow chart. If another re-evaluation is desired, do as explained above and put the new records behind the group of records for the first re-evaluation.

There is no limit to the number of re-evaluations. The time limits on the computer may have to be increased if the user has a large number of re-evaluations.

Basicly the program reads input records until there is a "1" in column one of an input record or until the end of the file is reached. The first input record that contains a "1" in column one will flag the program that the set of matrices derived from the flow chart have been built and are ready to be taken through the transformation process. After the matrices have been transformed by the parallel equations the information in the matrices is stored in three additional matrices. The information in the stored in three additional matrices. The information in the stored matrices will be used by the program for re-evaluation of the original matrices mentioned above. Once the transformation process is completed and the results printed, the original matrices are zero filled.

Next, the information on the input record containing the "1" in column one is put in the appropriate cells of the zero filled matrices and the re-evaluation procedure begins. Input records are continually read and the information is put in the matrices until there is another record with a "1" in column one or the end of file is reached. At this point the only information in the new matrices are changes in the probabilities and the weights associated with cells from the original matrices. Therefore, the values associated with cells that are not being re-evaluated must be placed in the appropriate cells of these matrices. The program accomplisher this by taking the information in the cells of the stored matrices and puts it in the appropriate cells that are zero filled within the new matrices. The matrices are then taken through the transformation process. The re-evaluation

procedure will be repeated as long as there are further input records read containing "l's" in column one.

If the end of file is reached after the building of the original matrices, the program stops after the matrices are taken through the transformation process. If the end of file is reached after the reading of re-evaluation input, the program stops after the re-evaluation procedure is completed.

# III. ENQUEUE Module Performance

The module ENQUEUE was selected as a typical, frequently-used RADACS module to be analyzed using the techniques of section 2. Attachment 9 displays the EUQUEUE flowchart. The source code was examined and each branch execution time was calculated. The resulting analysis transformed the original flowchart into the one of Attachment 10. Next, the control and data cards were prepared under the following

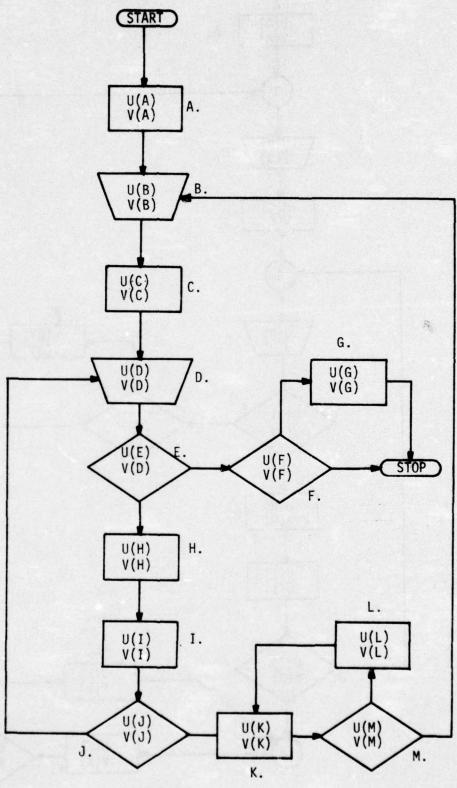
From Node	To Node	Probability Estimates	Measured Probability	Mean Execution (memory cycles)
1	2	.01	0.00	9
1	10	.99	1.00	23
10	9	.50	.998635	2
10	3	.50	.001365	5
3	9	.50	.58	9 7
3	4	.50	.42	10 years (10 to 10
4	5	.50	0.00	5 1
4	6	.50	1.00	5
5	4	.50	.50	11
5	6	.50	.50	5
6	8	.50	1.00	5
6	7	.50	0.00	1
7	9	.50	.71	8
7	9	.50	.29	19
8	8	.50	.50	7
8	9	.50	.50	8
9	2	1.00	11 1.00	2

First an estimate of the decision point probabilities was made by the module programmer. Then branch counters were inserted into the code to experimentally determine the same probabilities. Note that there was considerable difference in the human estimate and experimental values, as is reflected in the resulting analysis displayed below:

	Estimated Case	Experimental Results
Mean Execution in Machine Cycles	s 39.5	27.0
Variance	311.1	1.5

The experimental results are quite impressive. This analysis of the ENQUEUE module says that one can expect each execution of ENQUEUE to take 27-29 memory cycles, or 14-17 µsec.

The analysis technique presented here is well suited to analyzing all of the other RADACS modules. Then with each module reduced to a single block, higher levels of the operating system are then suitable for a similar analysis.



ATTACHMENT 1.

U(M) V(M)

M.

	CONTROL DECOR			
RECORD DESCRIPTION	RECORD POSITION	FORMAT	SPECIAL INSTRUCTIONS	LIMIT
Total Number of Nodes	001-003	13	Cannot be Left Blank	
Print Transformation of Original Flow Chart	004	п	"1" if This Option is Desired	
Print Transformation of the Re-evaluations of the Flow Chart	900	==	"1" if This Option is Desired	
Depth of Matrices	800-900	13	Cannot be Left Blank	003
Control Card Flag	600	A1	Must Contain an "2"	

# DATA RECORD

LIMIT		010	010			
SPECIAL INSTRUCTIONS	"1" if This Option Desired					
FORMAT	Π	13	13	F9.8	F10.5	F10.5
RECORD POSITION	100	002-004	005-007	008-016	017-026	027-036
RECORD DESCRIPTION	Re-evaluation Flag	Excident Node of the Link	Incident Node of the Link	Probability of the Link	Mean of the Link	Variance of the Link

# Parallel Equations

Probability 
$$P_{ij} = \sum_{n} P_{ij;n}$$

Mean 
$$\mu_{ij} = \frac{\sum_{n}^{\mu} \mu_{ij;n} \cdot P_{ij;n}}{\sum_{n}^{\mu} P_{ij;n}}$$

Variance

$$\lambda_{ij} = \frac{\sum_{n}^{\sum_{i} \lambda_{ij;n} \cdot P_{ij;n}}}{\sum_{n}^{\sum_{i} P_{ij;n}}} + \frac{\sum_{n}^{\sum_{i} \mu_{ij;n}^{2} \cdot P_{ij;n}}}{\sum_{n}^{\sum_{i} P_{ij;n}}} - \frac{\left[\sum_{n}^{\sum_{i} \mu_{ij;n} \cdot P_{ij;n}}}{\sum_{n}^{\sum_{i} P_{ij;n}}}\right]^{2}$$

Loop Equations

Probability 
$$P_{ij} = \frac{P_{ij}'}{1 - P_{ii}}$$

Mean 
$$\mu_{ij} = \mu'_{ij} + \frac{\mu_{ii}^{p}_{ii}}{1 - P_{ii}}$$
Variance 
$$\lambda_{ij} = \lambda'_{ij} + \frac{\lambda_{ii}^{p}_{ii}}{1 - P_{ii}} + \frac{\mu^{2}_{ii}^{p}_{ii}}{(1 - P_{ii})^{2}}$$

The Cross Term Equations

Probability 
$$P_{ik} = P_{ij} \cdot P_{jk}$$

Mean 
$$\mu_{ik} = \mu_{ij} + \mu_{jk}$$

Variance 
$$\lambda_{ik} = \lambda_{ij} + \lambda_{jk}$$

ATTACHMENT 5. [1]

```
DIMENSION DF(10,10,1), DU(10,10,1), DSIG(10,10,1)
     COMMON/Y1/P(10,10,3)/Y2/U(10,10,3)/Y3/SIG(10,10,3)
     INTEGER A
     REAL MN, MEANE
      READ(8,10,END=800)
                           MTRXSZ, MTRXP, LVARC, MDPTH, A
     IF(A.NE.2)GO TO 500
     N=MTRXSZ
     M=MDPTH
     IF (M.EQ.0)GO TO 840
     MSTRE=1
     LHEDR=1
     LEVAL=0
     DO 20 I=1,N
     DO 20 J=1,N
     DO 20 K=1.M
     P(I,J,K)=0
     U(I,J,K)=0
     SIG(I, J, K)=0
20
     CONTINUE
22
     READ(8,24,END=820) LCT,LR,LC,P1,U1,SIG1
     IF(LCT.EQ.1)GO TO 29
     DO 26 KK=1,M
     IF(F(LR, LC, KK), EQ.O.)GO TO 28
26
     CONTINUE
28
     P(LR, LC, KK)=P1
     U(LR, LC, KK)=U1
     SIG(LR, LC, KK) = SIG1
     GO TO 22
29
     IF (MSTRE.EQ.1)50 TO 38
     IF(LVARC.EQ.1)GO TO 32
     MTRXP=2
     GO TO 34
32
     MTRXP=1
34
     DO 36 I=1,N
     DO 36 J=1,N
     IF(P(I,J,1).NE.0.)GD TO 36
     P(I,J,1)=DP(I,J,1)
     U(I,J,1)=DU(I,J,1)
     SIG(I,J,1)=DSIG(I,J,1)
     CONTINUE
36
38
     DO 44 IS=1,N
     SUM=0.
     DO 40 JS=1,N
     DO 40 KS=1,M
     SUM=SUM+P(IS,JS,KS)
40
     IF(IS.EQ.2)GO TO 42
     IF(SUM.NE.1.)GO TO 880
     GO TO 44
42
     IF(SUM.NE.O.)GO TO 860
44
     CONTINUE
     DO 48 LS=1,N
     SUM=SUM+P(N,LS,1)
48
     CONTINUE
```

```
IF (SUM.EQ.0.)GO TO 900
55
     II=N
     JJ=N
     CALL PARALL(II, JJ, P, U, SIG, M)
     IF (MTRXP.NE.1)GO TO 86
     IF (LHEDR.NE.1)GO TO 81
     IF(LEVAL.NE.O)GO TO 70
     WRITE(5,855)
     GO TO 73
70
     WRITE(5,856) LEVAL
73
     LHEDR=2
     DO 84 I=1,N
81
     DO 82 J=1,N
     WRITE(5,836) N,N,P(I,J,1),U(I,J,1),SIG(I,J,1),I,J
82
     CONTINUE
     WRITE(5,875)
84
     CONTINUE
     IF (MSTRE.NE.1)GO TO 88
86
     DO 87 I=1,N
     DO 87 J=1,N
     DP(I_{J}J_{J}1)=P(I_{J}J_{J}1)
     DU(I,J,1)=U(I,J,1)
     DSIG(I,J,1)=SIG(I,J,1)
87
     CONTINUE
     MSTRE=2
88
     DO 89 I=1,N
     IF(P(I,I,1).EQ.0.)GO TO 89
     II=I
     CALL LOOP(II, JJ, P, U, SIG, M)
89
     CONTINUE
     IF(N.EQ.2)GD TO 90
     CALL CRSTRM(JJ,JJ,P,U,SIG,M)
     N=N-1
     M=2
     GO TO 55
90
     PRBLTY=F(1,2,1)
     MEANE=U(1,2,1)
     VARNCE=SIG(1,2,1)
     IF(LEVAL.NE.O)GO TO 92
     WRITE(5,600) PRBLTY, MEANE, VARNCE
     GO TO 93
92
     WRITE(5,620) PRBLTY, MEANE, VARNCE, LEVAL
93
     IF(LCT.EQ.2)GO TO 100
     LHEDR=1
     N=MTRXSZ
     M=MDPTH
     DO 95 I=1,MTRXSZ
     DO 95 J=1,MTRXSZ
     DO 95 K=1,MDPTH
     P(I,J,K)=0
     U(I_{\gamma}J_{\gamma}K)=0
     SIG(I,J,K)=0
95
     CONTINUE
```

```
LEVAL=LEVAL+1
     P(LR,LC,1)=P1
     U(LR, LC, 1)=U1
     SIG(LR,LC,1)=SIG1
     GO TO 22
500
     WRITE(5,520)
     GO TO 100
800
     WRITE(5,810)
     GO TO 100
820
     LCT=2
     GO TO 29
840
     WRITE(5,850)
     GO TO 100
860
     WRITE(5,870)
     GO TO 100
     WRITE(5,890)
880
                    IS
     GO TO 100
900
     WRITE(5,920)
                    MTRXSZ
     GO TO 100
100
     STOP
10
     FORMAT(13,11,11,13,11)
     FORMAT(I1,213,F9.7,2F10.5)
24
520
     FORMAT(10X,49HMISSING PARAMETER CARD OR PARAMETER CARD IN ERROR.
    X//)
    FORMAT(10X,14HPROBABILITY = ,F9.7,10X,7HMEAN = ,F10.5,10X,
600
    X11HVARIANCE = ,F12.5,05X,29HRESULT OF ORIGINAL FLOW CHART,//)
     FORMAT(10X,14HPROBABILITY = _{7}F9.7,10X,7HMEAN = _{7}F10.5,10X,
    X11HVARIANCE = ,F12.5,05X,23HRESULT OF RE-EVALUATION,2X,13,//)
810
     FORMAT(10X,15HNO DATA RECORDS,//)
830
     FORMAT(213, F9.7, 2F10.5)
835
     FORMAT(213, F9.7, 2F10.5)
834
     FORMAT(34X,13,5H BY ,13,2X,F10.8,3X,F10.5,2X,F10.5,2X,13,4X,13,
    X38X)
850
     FORMAT(10X,25HDEPTH OF MATRIX NOT GIVEN,//)
     FORMAT(2X,35HCOMPUTATION OF INITIAL FLOW CHART ,6HMATRIX,3X,
    X11HPROBABILITY, 6X, 4HMEAN, 6X, 8HVARIANCE, 3X, 3HROW, 2X, 6HCOLUMN, 37X,
    X//)
     FORMAT(3X,29HCOMPUTATION OF RE-EVALUATION ,13,2X,6HMATRIX,3X,
    X11HPROBABILITY, 6X, 4HMEAN, 6X, 8HVARIANCE, 3X, 3HROW, 2X, 6HCOLUMN, 37X,
    X//)
870
     FORMAT(10X,41HINPUT NOT BUILT CORRECTLY FROM FLOW CHART,
    X37H, ROW 2 SHOULD NOT CONTAIN ANY VALUES,//)
875
     FORMAT(1HO)
     FORMAT(10X,27HSUM OF PROBABILITIES IN ROW,13,
    X17H NOT EQUAL TO ONE,//)
     FORMAT(10X,41HMATRIX IN ERROR NO INFORMATION IN COLUMN ,13,//)
```

END

SUBROUTINE LOOP(II, JJ, P, U, SIG, M) DIMENSION F(10,10,3),U(10,10,3),SIG(10,10,3) COMMON/Y1/PRB(10,10,1)/Y2/MN(10,10,1)/Y3/VAR(10,10,1) REAL MN DO 30 J=1,JJ IF(II.EQ.J)GO TO 30 IF(F(II,J,1).EQ.0.)GO TO 30 PRB(II,J,1)=P(II,J,1)/(1-P(II,II,1)) MN(II,J,1)=U(II,J,1)+(U(II,II,1)\*P(II,II,1))/(1-P(II,II,1)) VAR1=SIG(II,J,1)+(SIG(II,II,1)\*P(II,II,1))/(1,-P(II,II,1)) VAR2=(U(II,II,1)\*\*2\*P(II,II,1))/(1-P(II,II,1))\*\*2 VAR(II,J,1)=VAR1+VAR2 CONTINUE PRB(II, II, 1)=0 MN(II,II,1)=0 VAR(II,II,1)=0 RETURN END

30

SUBROUTINE CRSTRM(II, JJ, P, U, SIG, M)
DIMENSION P(10,10,3), U(10,10,3), SIG(10,10,3)
COMMON/Y1/PRB(10,10,1)/Y2/MN(10,10,1)/Y3/VAR(10,10,1)
REAL MN
III=II-1
DO 40 I=1,III
DO 40 K=1,III
IF(P(I,K,1).EQ.0.)GO TO 32
L=2
GO TO 34
L=1
PRB(I,K,L)=P(I,JJ,1)\*P(JJ,K,1)
IF(PRB(I,K,L).EQ.0.)GO TO 40
MN(I,K,L)=U(I,JJ,1)+U(JJ,K,1)

VAR(I,K,L)=SIG(I,JJ,1)+SIG(JJ,K,1)
40 CONTINUE
RETURN
END

32

34

SUBROUTINE PARALL(II, JJ, P, U, SIG, M) DIMENSION F(10,10,3),U(10,10,3),SIG(10,10,3) COMMON/Y1/FRB(10,10,1)/Y2/MN(10,10,1)/Y3/VAR(10,10,1) REAL MN, NMU INTEGER D DO 400 I=1,JJ DO 400 J=1,JJ FSUM=0 NMU=0 V1N=0 V2N=0 IF(P(I,J,1).EQ.0.)GD TO 400 IF(P(I,J,2).EQ.0.)GO TO 400 DO 270 L=3,M IF(P(I,J,L).EQ.0.)GD TO 280 270 CONTINUE 280 LL=L-1 DO 300 D=1,LL PSUM=PSUM+P(I,J,D) NMU=NMU+U(I,J,D)\*F(I,J,D) V1N=V1N+SIG(I,J,D)\*P(I,J,D) V2N=V2N+(U(I,J,D))\*\*2\*P(I,J,D) 300 CONTINUE MN(I,J,1)=NMU/PSUM VAR(I,J,1)=(V1N/PSUM)+(V2N/PSUM)-(NMU/PSUM)\*\*2 PRB(I,J,1)=PSUM CONTINUE RETURN END

INPUT:					
004000012		CONTR	OL CARD)		
0010031.0	20.1	31.6	*		
0030020.2	15.2	28.3	*		
0030040.8	10.4	16.4	*		
0040011.0	8.3	13.3	END OF ORIGIN	IAL DATA	
10030020.3	16.1		*		
0030040.7	12.0		END OF RE-EVA	LUATION 1	
10040011.0	22.3		END OF RE-EVA	LUATION 2	
OUTPUT:					
PROBILITY	= 1.000000	DO MEA	N = 190.50000	VARIANCE	=30413.89917
	RESULT	OF ORIG	INAL FLOW CHART		
PROBILITY	= 1.000000	DO MEA	N = 130.46667	VARIANCE	=12830.94470
	RESULT	OF RE-E	VALUATION 1		
PROBILITY	= 1.000000	OO MEA	N = 246.50000	VARIANCE	=56008.69727
	RESULT.	OF RE-F	UALHATTON 2		

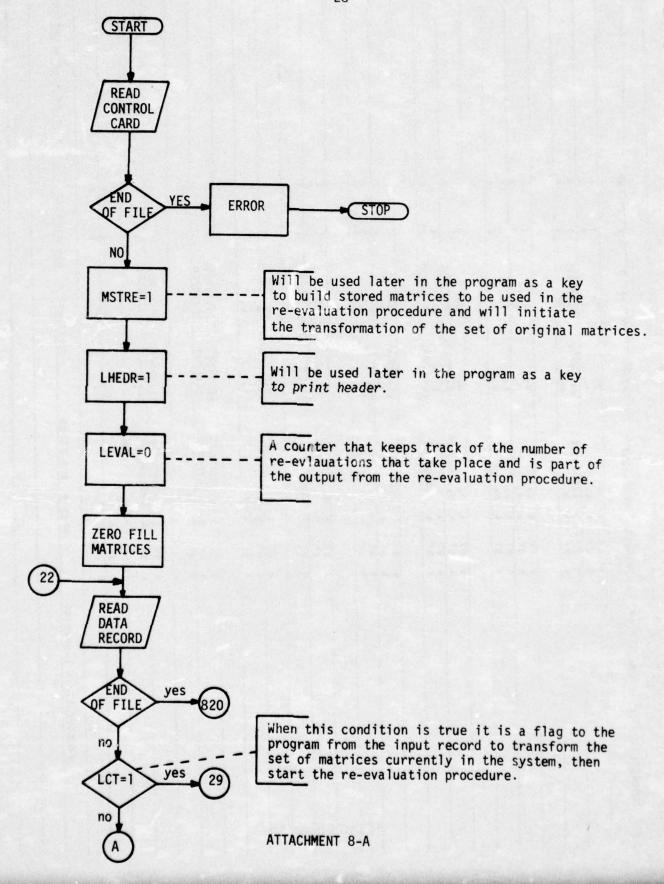
INPUT:					
004110012		CONTR	OL CARD	)	
0010031.0	20.1	31.6	*		
0030020.2	15.2	28.3	*		
0030040.8	10.4	16.4	*		
0040011.0	8.3	13.3	END O	F ORIGINAL DATA	A
10030020.3	16.1		*		
0030040.7	12.0		END O	F RE-EVALUATIO	N 1
10040011.0	22.3		END O	F RE-EVALUATIO	N 2

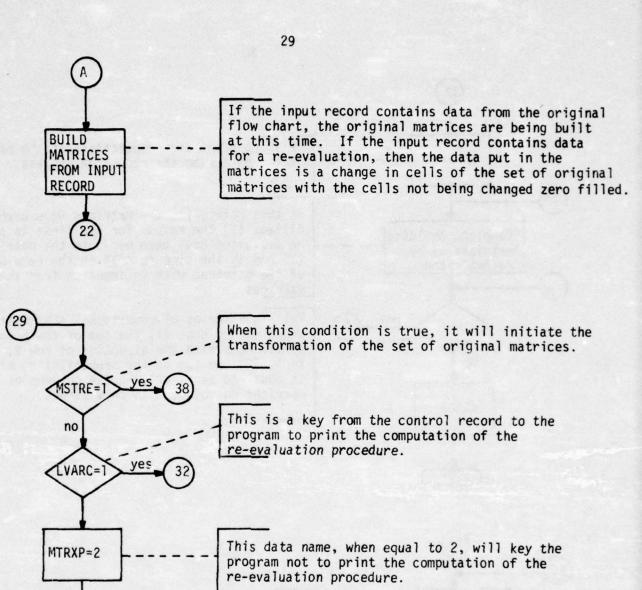
**OUTPUT: (SEE NEXT 3 ATTACHMENTS)** 

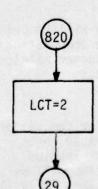
BY 4 0.80000000000000000000000000000000000
--

																46								RESULT OF RE-EVALUATION	
COLUMN			m +	-	N 10	•	-	2		•	2 .			- 2	1 10	- •	N P	1		-	~		2		
ROW	1	-	- -	~	<b>~</b> ~	2	-	,	20		• •	•	.	-		~	~	-	, ,	-	-	2	2	12830.04470	
VARIANCE		• 0	31.60000	•	• •	٥.	••	•		13.30000	•			.00	31.60000	•	.0	13.30000	:	44.90000	31.60000		• 0	VARIANCE = 12	
MEAN		.0	00001.0000	•	• •	•0	•	16.10000	12.00000	6.30000	•			.00	20.10000			20.30000	•	40.40000	36.20000		• 0	AX.	
PROBABILITY		•	1.000000000	.0		•	9.0	0.000000	0.10000000	000000000	••				1.0000 000	• • •		0.70000000	•	0.76655666	0.30000000	•	•	130.461.87	
											• •	1	1			-	1	1	•		~	~ .	1	HEAN .	
MATRIX	*	1		2			*	-		*		-					1				2	*			
COMPUTATION OF RE-EVALUATION 1																								PROBABILITY = 1.0000000	

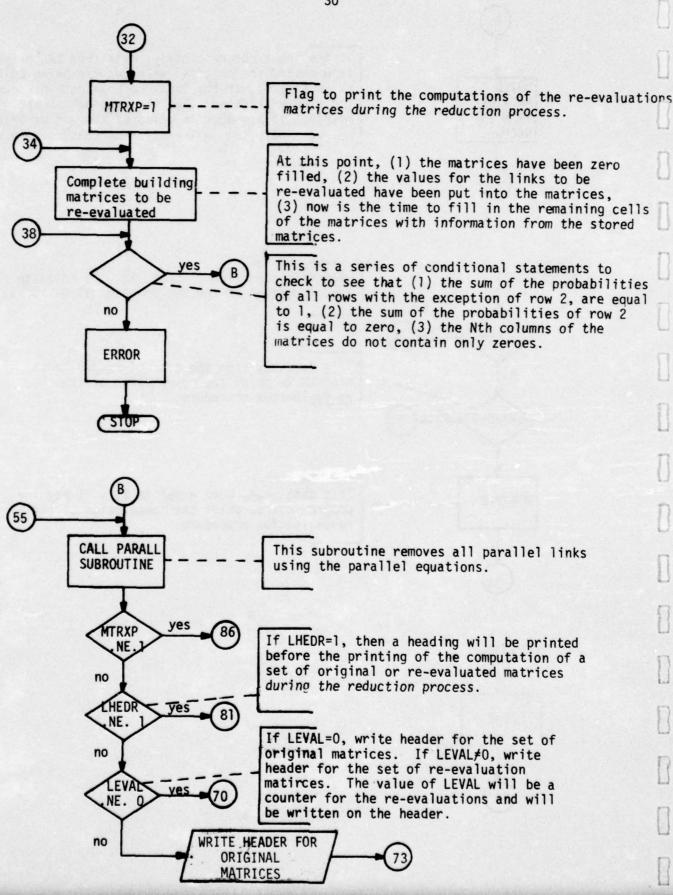
													27									RESULT OF RE-EVALUATION 2
21	1 2	me	-	2	, 4		3	4	- 8	ne		2			3	1 2	3	-	2	•	2	
ROW COLUMN				2	. 2	8	5 5	8		44	-	- -		2	2	m m	n	1	-	2	2	66008.69727
VARIANCE		31.60000	• 0	•	•	•0	28.30000	16.40000		••	•	31.60000			6	16.40000	• 0	48.00000	29.90000	• 0	• 0	VARIANCE
HEAN		20.10000		•00	•	•0	15.20000	10.40000	22.30000	••	• 0	20.10000			•0	32.70000	•	52.80000	25.30000	•	•0	877
WALET TY	1	1.0000000000000000000000000000000000000					0.2000000	0.00000000	1.000000000			1.00000000				0.80000000		0.00000000	0000000			246.50000
	00	-0		1		0	1				3 0.	1		30.		3 0.2		2 0.8	1		2 0.	MEAN .
MATRIX	25	* *			-	>	9		**		2	1			1	> N N		> 0 0		× 8	, Q 8	
COMPUTATION OF RE-EVALUATION .																						PROBABILITY = 1.00000000
DUTATION O		=																				PROBA
CON										A T.	АСНМ	-	T 71									

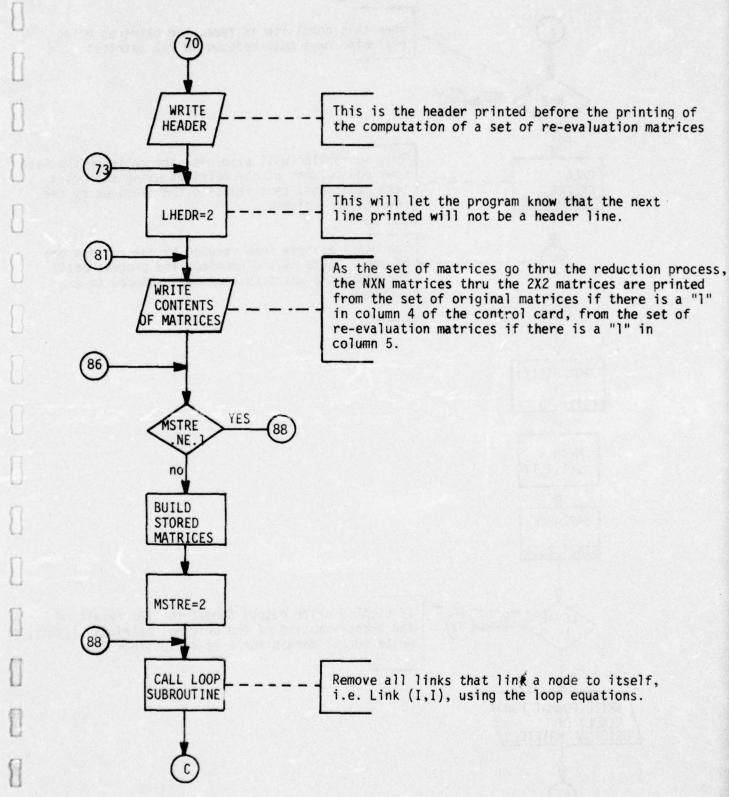




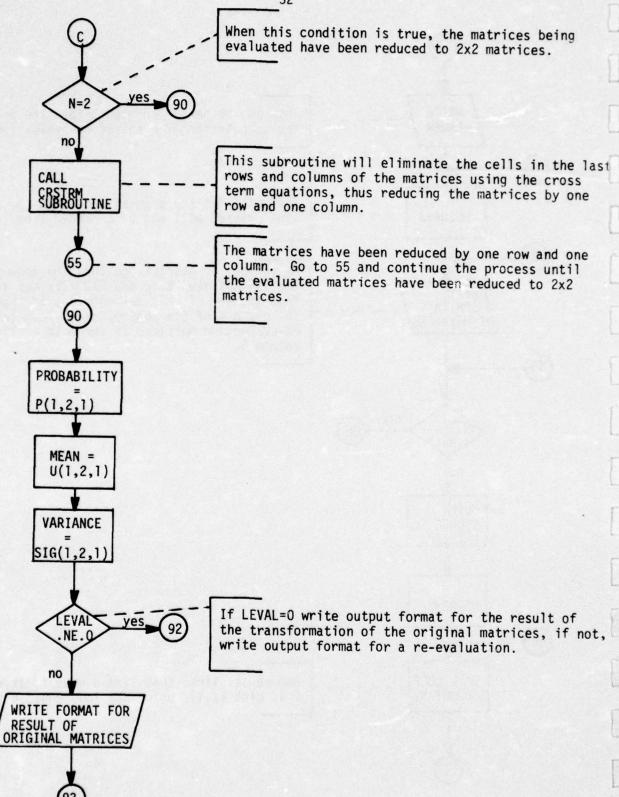


ATTACHMENT 8-B

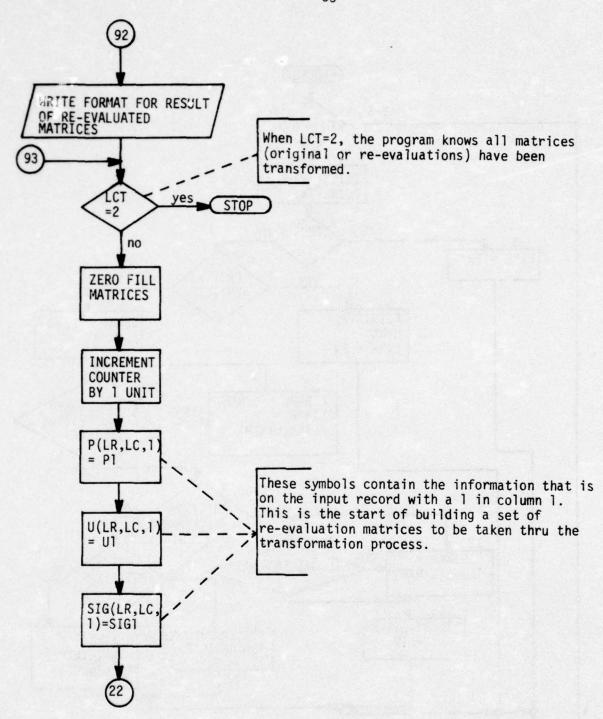




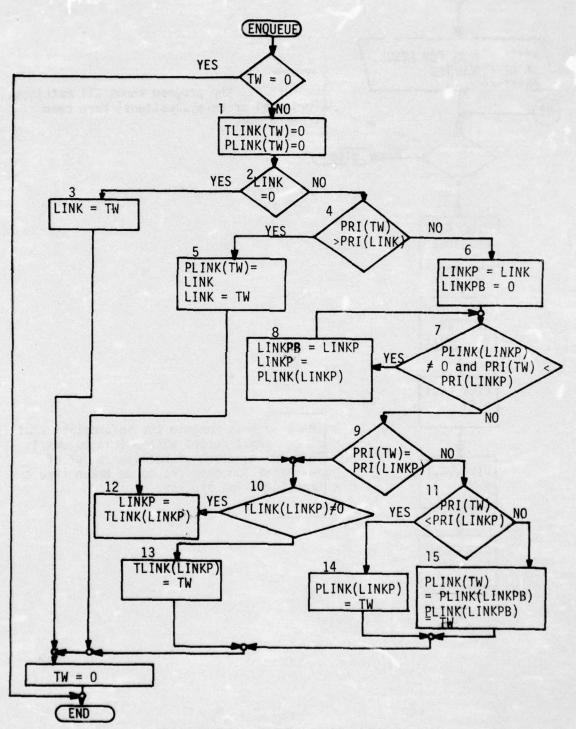
ATTACHEMENT 8-D



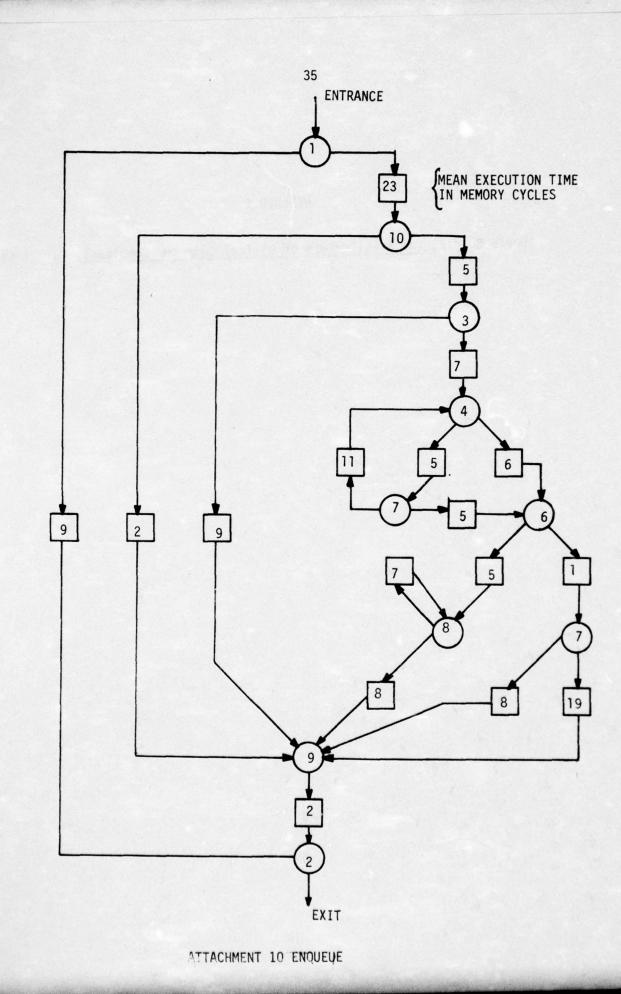
ATTACHMENT 8-E



ATTACHMENT 8-F



ATTACHMENT 9: FLOWCHART OF ENQUEUE



# REFERENCE

Boris Beizer, The Architecture of Digital Computer Complexes, Vol. 1 and 2.

PART FOUR

GTE-SYLVANIA COLOR CRT

PROGRAMMER'S REFERENCE MANUAL

for

U. S. Army Missile Command Redstone Arsenal, Alabama

by

Engineering Experiment Station
Auburn University
Auburn, Alabama
Under Contract DAAH01-71-C-1303

Prepared by:

J. H. McElreath Graduate Research Assistant H. A. Jackson Graduate Research Assistant

Reviewed by:

H. T. Nagle, Jr. Associate Professor

Approved by:

J. D. Irwin Associate Professor and Head

# Table of Contents

Section I.	General Description
	A. General Information
Section II.	Operation
	A. Controls for the CRT
Section III.	Programming
	A. Instruction Set
	3. Color Changing C. Input Programming
	D. Interrupt Processing

## GTE Sylvania Color CRT Programmer's Reference Manual

# Section I. General Description

#### A. General Information

The GTE Sylvania Model CD-2130B (Modified) is a four color CRT display used for operator communications in the EAR/RADACS system.

The CRT will display signals returned from the radar system, text generated by the user and computer, map overlays, and other information useful or pertinent to a radar control display.

# B. Functional Description

A functional diagram for the color CRT display system is given in Figure 1.

Functionally, the SEL-86 communicates with the CRT and user through the Device Controller Channel (DCC). The DCC, in turn, "talks" to the Serial Data Terminal (SDT), located in the CRT cabinet. The SDT handles all data communications for the CRT.

Data from the computer or radar are input from the SDT to the Display Generator (DG). The DG then sends the data to the proper output section for display.

To display a vector, the following information must be supplied:

the initial x and y coordinates and the final x and y coordinates, or some
combination of old and new coordinates with the "last transfer" bit set on
the last data item (more on this is given in the section on Programming).

This information is sent to the Vector Generator which controls the deflec-

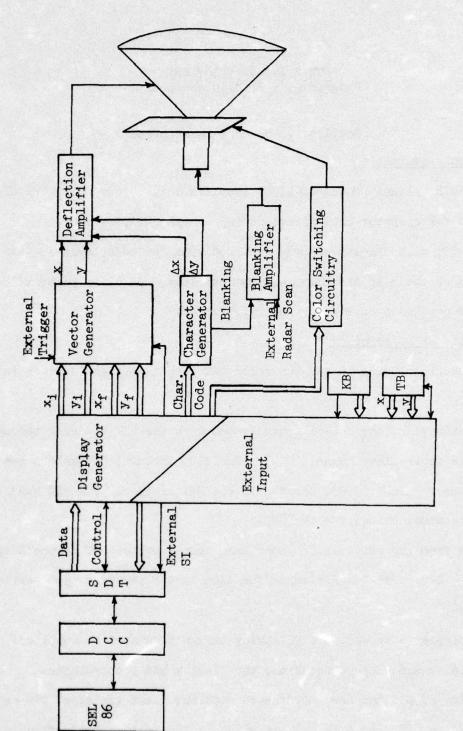


Figure 1. Functional Diagram of CRT.

tion amplifier and draws the vector.

Similarly, character information must include the character code and the x and y position of the center of the character. This information is sent to the Character Generator which controls the deflection amplifier and blanking amplifier and thus draws the character.

The color switching circuitry is composed of a high voltage switch which changes between outputs of 6-, 8-, 10-, and 12 kilovolts to produce a red, orange, yellow or green display output, respectively, as desired.

The CRT uses a beam penetration method for producing colors with a 2-layer phosphor screen (see Figure 2.) The degree of penetration into the phosphors by the beam determines the color on the screen.

The display can produce 2 kinds of vectors - those drawn automatically and those drawn by external triggering and modulation. Vectors drawn automatically are ones from the computer where the initial and final points are supplied and the vector is drawn accordingly. Those drawn using the external trigger are given the initial and final points from the computer, but the vector is not drawn until it receives an external trigger. Further, the externally triggered vector is drawn by external modulation of the beam drawing the vector. This is useful in the present radar application in that the computer can "tell" the phased array radar where to put a beam, then "tell" the display where to draw the corresponding vector. The returning beam can then intensity modulate the beam on the display directly (the rate of drawing the vector will be synced to the return of the beam). How this is done is shown in the section on Programming.

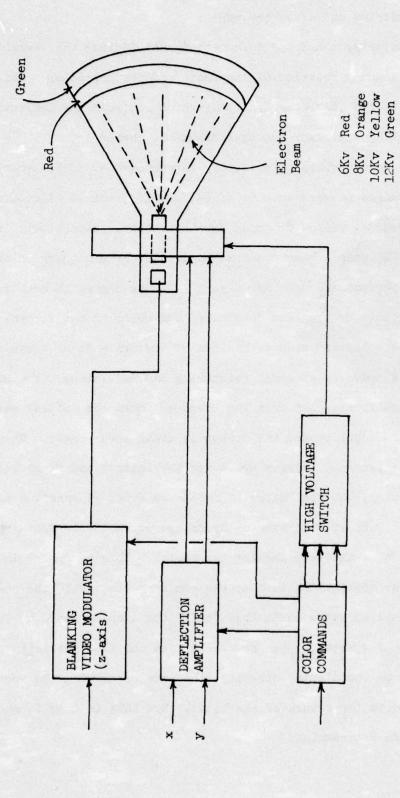


Figure 2.

Council

Information from the keyboard (characters) and trackball (an x and y position) is handled by external input circuitry which generates an external interrupt to be sent to the computer. The computer answers, gets the information (via the DCC and SDT) and uses it as necessary.

The trackball is used to control the position of the cursor on the screen. Hardware within the display unit keeps track of the position of the cursor but the software (programmer) decides whether it is to be displayed or not.

The TRACKBALL ENABLE key sends an interrupt to the computer requesting service. If the interrupt handler for it is enabled and if the computer is programmed to respond to it, various types of position information may be entered. This information may be an initial or final point of a vector, it may designate a target to be tracked, or whatever the programmer desires. The TRACKBALL HOME key brings the cursor back to the "home" position in the lower left corner of the screen. (See Figures 3 and 4).

The keyboard has the standard array of alphanumeric and punctuation characters plus 5 keys on the left and 15 keys on the right which correspond to the special characters in the ASCII code. (See Figure 4). Any key may be defined to be used in any way. However, those on the standard keyboard are usually defined as they normally would be (letters, digits, etc.) and the special function keys defined in any way the programmer wishes. For example, some of these keys may be designated to handle data or commands such as initial and final vector position, "track target designated by the trackball position," etc.

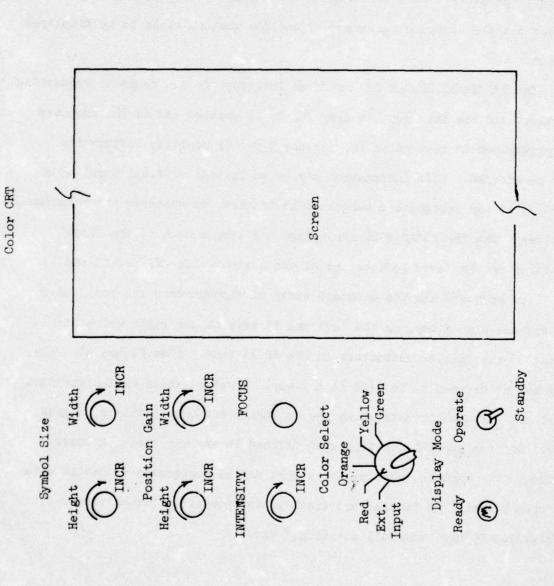


Figure 3

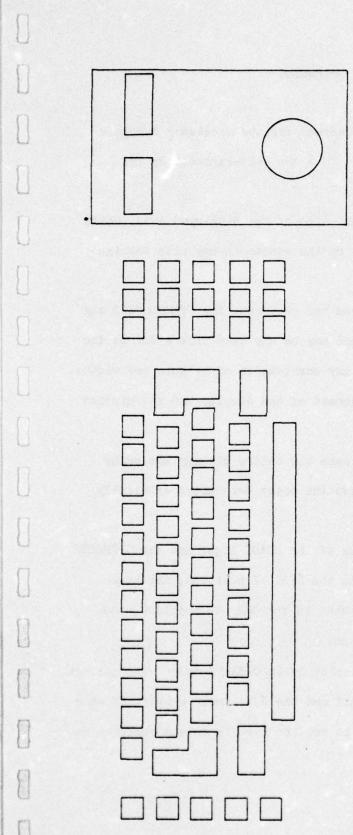


Figure 4

# Section II. Operation

# A. Controls for the CRT

To the left and right of the CRT screen are the necessary controls to adjust the display (see Figure 3). They are self-explanatory for the most part.

POSITION GAIN controls the overall size of the displayed area, which may be from full screen down to a dot on the screen in any size combination of both height and width.

After the size of the display area has been set, the SYMBOL SIZE may be adjusted within this area. A symbol may be any size from a dot on the screen up to about an inch square in any combination of heights and widths.

INTENSITY sets the overall brightness of the display and the picture may be FOCUSed as necessary.

COLOR SELECT allows the user to make the entire display one color (red, orange, yellow, or green) or have the color determined externally under program control.

The DISPLAY MODE section consists of the READY light and the OPERATE/
STANDBY switch. When the switch is in the STANDBY position, the high
voltage switch power supply is off. When in the OPERATE position, the
H.V. power supply is on and ready to go.

The READY light indicates the display is in OPERATE mode. The cabinet has interlocks, however, to keep it off and the H.V. power supply off when the cabinet is open. When the light is on, the tube is on and ready to go.

# B. Programmed I/O

The CRT display is programmed as though it were a 1024 x 1024 position matrix, referenced to the lower left corner. It has a 21 inch diameter with a usuable area of  $11 \times 15$  inches.

Vectors may be drawn from any point to any other point. Characters may be input from the keyboard and placed at any position on the screen. The cursor may be placed at any position, determined by the trackball, and the position of the trackball/cursor may be read into the computer at the programmer's request (hardware keeps up with where the trackball is and the software displays it).

Any character, vector, or the cursor may be displayed in any of the 4 colors desired.

Vectors may be displayed either by the automatic trigger or by an external trigger, as described earlier with details given in the Programming section.

Things to be displayed are put in a list in consecutive locations in memory. A Command Device (CD) instruction is used to initiate the transfer to the display. When the transfer is complete, a Service Interrupt (SI) occurs to let the computer know it is ready for more input, if desired.

An SI will also occur if a keyboard button or a trackball button is pressed, after which the computer will acknowledge the request and pick up the data, under program control, if it so desires.

The Test Device (TD) instruction is used to determine the status of the I/O channels and devices.

To prevent the display from flickering, the picture must be refreshed

about 50 times each second. The picture may be refreshed by the program if, for example, calculations are to be done between each frame to change the displayed information. The picture may simply be redrawn each time, as in the case of using the external interrupt and external trigger when directly displaying radar "images," as explained earlier. The display may also be refreshed automatically by using the interval timer to send an interrupt to the computer to cause it to send the Command Device and data to the display at regular intervals.

A more complete description and list of capabilities are given in the section on Programming.

# SECTION III: Programming

### A. Instruction Set.

Actual control of the CRT is initiated through the use of a COMMAND DEVICE instruction. 'Upon execution of this instruction, data is transferred either from core to the display unit or from the display unit to core. Both the address and amount of data is determined by the TCW (TRANSFER CONTROL WORD), which is located at location X'120' in core. The TCW along with the COMMAND DEVICE instruction are described below:

(	) :	1	2	3	4	5	6	7	8	9	10	11	12	13	1	+1	516 17 18 19 20 21 22 23 24 25 26 27 28 29 30	31
6		1	1	1	1	1	1	0	0	0	0	0	0	1	1	0	FUNCTION SI TX TY K 0 0 0 0 0	

bits 0-15: always X'FEO6' for display unit.

bits 16-19: X'B' for input, X'A' for output.

bit 20 : reset EXTERNAL SERVICE INTERUPT.

bit 23 : 0 for output, indicates "read trackball x co-

ordinate" for input.

bit 24 : 0 for output, indicates "read trackball y co-

ordinate" for input.

bit 25 : 0 for output, indicates "read keyboard" for input.

bits 26-31: always off for display unit.

TRANSFER CONTROL WORD (TCW)

bits 0-11: number of half-words of data to be transferred to

or from display unit.

bits 12-31: memory location of data to be transferred.

It is important to note that only one halfword of data may be transferred for the input type of CD instruction, except for the "read track-ball" type of input CD, which transfers two halfwords. Thus the TCW will contain a transfer count of one for all input CD's except when the track-ball is read, in which case the count will be two. As would be expected, upon execution of an input type of CD instruction, data is transferred from the display unit to the address specified by the TCW.

When an output type of CD instruction is executed, data is transferred from the location specified by the TCW to the display unit. The display unit examines the data as it needs it and treats each half-word of data as either 1)control information, 2) one of the co-ordinates of a vector (or character), or 3) a character to display. The particular type of data is determined by the first three bits of each halfword which may be called the identifier for that particular halfword. The various identifier codes are listed below:

IDENTIFIER FUNCTION (Bits 0,1,2)

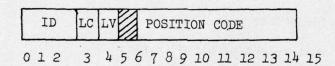
0

illegal

1	initial x yalue
2	initial y value
3	final x value
4	final y value
5	display character
6	control information
7	illegal

All data can be classified as one of three types:

TYPE 1) Identifier codes of 1, 2, 3, or 4.



bits 0-2 : Identifier code.

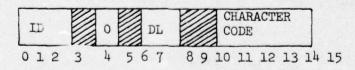
bit 3 : character last transfer.

bit 4 : vector last transfer.

bit 5 : unused.

bits 6-15: x or y co-ordinate (0-1023)

TYPE 2) Identifier code of 5.



bits 0-2 : Identifier code.

bit 3 : Unused.

bit 4 : 0

bit 5 : Unused

bits 6-7: Delay code (00-maximum delay, 11-minimum delay).

bits 8-9 : Unused.

bits 10-15: 6-bit ASCII character code for character to be displayed.

# TYPE 3) Identifier code of 6.



bits 0-2 : Identifier code

bit 3 : Must be set if bit 7 is set.

bit 4 : Vector last transfer bit.

bits 5-6 : Unused.

bits 7 : Display cursor

bits 8-9 : Unused.

bit 10 : Enables change of trigger source

bit 11 : Reset for external trigger, set for internal trigger.

bit 12 : Unused

bit 13 : Enables color change

bits 14-15: If bit 13 is set, a two bit color code is used:

00 - Red

01 - Orange

10 - Yellow

11 - Green

Sometimes it may be necessary to test the display unit to determine information about it. To accomplish this task, a TEST DEVICE (TD) instruction is available:

# TEST DEVICE (TD)

111111	1000000 101 TEST CODE		
	6 7 8 9 10 11 12 131415 16	27 28 29	30 31

bits 16-31 : Type of test desired:

X'8000' - Sets condition code according to following:

ccl - Delay (retest)

cc2 - Channel active

cc3 - DCC error (test with code X'4000')

cc4 - Device abnormal (Test with code X'2000')

X'4000' - Sets condition code according to following:

ccl - Invalid memory access

cc2 - Memory parity error

cc3 - Program violation

cc4 - Underflow or overflow

X'2000' - Sets condition code according to following:

cc2 = 0, Status transfer was performed.

cc2 = 1, Status transfer was not performed due

to DCC being active.

cc4 = 1, Device is absent or powered off.

If a TD with a test code of X'2000' is executed and  $cc2 = \emptyset$ , then the status bits have been set in the half-word which is pointed to by the address field in the TCW. These bits are useful in the handling of interrupts:

#### STATUS BITS



bit 1 - program violation

bit 2 - external service interrupt

bit 6 - indicates that keyboard set bit 2

bit 7 - indicates that TRACKBALL ENTER button set bit 2

bit 9 - set if illegal code received

bit 10 - set if color switching rate is too high

bit 11 - external trigger not received

This instruction is ordinarily used in the processing of interrupts. For example, if the operator wishes tracking information or wishes to communicate with the control program, the fact that an external service interrupt has occurred and what caused it (keyboard or trackball ready) is relayed to the control program through the status bits.

# B. Output Programming.

Output programming consists of the generation of characters or vectors onto the display unit and the changing and mixing of colors.

#### 1. Character Generation.

To generate a character onto the display unit, the type of character and its position on the screen must be transmitted. This can be accomplished by setting up a data list in core which consists of three halfwords: one to indicate the x co-ordinate of the character centroid, one to indicate the y co-ordinate of the character centroid,

and one to indicate which character is to be displayed. A CD instruction is then executed which transfers this data to the display unit, which then displays the particular character at the position desired. If several characters are to be displayed, this list may be extended and the appropriate characters will be displayed.

An added freature on the display unit allows the programmer to repeat the last character transferred without specifying the character again. To accomplish this feat, the programmer simply specifies the position for the character and turns the "character last transfer" bit on in the "x initial", "y initial", or "x final" data halfword. This process may be repeated as many times as necessary. Example program l illustrates the generation of characters and the "character last transfer" feature. It might be noted that the cursor is a character that can be generated by normal character generation. However, the hardware of the display unit is equipped to maintain the cursor position. Therefore, to display the cursor, the programmer must only set bit 7 of a data word to be transferred to the display unit that has an identifier code of 6. The unit will then display the cursor in the correct place.

#### 2. Vector Generation.

To generate vectors a little more information is needed to accomplish the task. In addition to the "x initial" and "y initial" data halfwords, a "x final" and a "y final" data halfword are needed. Upon the execution of a CD instruction that transfers this data list, in the order in which they are presented above, a vector will be drawn from the  $(x_i, y_i)$  coordinate, to the  $(x_f, y_f)$  co-ordinate.

```
PROGRAM
                     CHAREX
DISPUNIT EQU
                     X'40'
          DEFINE
*
          THIS SECTION IS A COLLECTION OF MACROS THAT SET UP DATA FOR
          THE DISPLAY UNIT
XINIT
          DEFM
                    POS, TRANS
          BOUND
                    1H
          IFT
                    C'XTRANS' .EQ. C'LC', ENDLC
         GEN
                    3/1,2/2,1/0,10/%POS
          GOTO
                    TRANFRES
ENDLC
          IFT
                    C'%TRANS' .EQ. C'LV', ENDLV
          GEN
                    3/1,2/1,1/0,10/%POS
          GOTO
                    TRANPRES
         ANOP
ENDLY
          GEN
                    3/1,2/0,1/0,10/%FOS
TRANFRES EXITM
          ENDM
                    POS, TRANS
YINIT
          DEFM
          BOUND
                    1H
          IFT
                    C'%TRANS' .EQ. C'LC', ENDLC
         GEN
                    3/2,2/2,1/0,10/%POS
         GOTO
                    TRANPRES
ENDLC
         IFT
                    C'%TRANS' .EQ. C'LV', ENDLU
         GEN
                    3/2,2/1,1/0,10/%POS
         GOTO
                    TRANPRES
ENDLY
         ANOF
         GEN
                    3/2,2/0,1/0,10/%FOS
TRANPRES EXITM
         ENDM
XFIN
         DEFM
                    POS, TRANS
         BOUND
                    14
          IFT
                    C'%TRANS' .EQ. C'LC', ENDLC
         GEN
                    3/3,2/2,1/0,10/%POS
         GOTO
                    TRANPRES
ENDLC
         IFT
                    C'XTRANS' .EQ. C'LV', ENDLV
         GEN
                    3/3,2/1,1/0,10/%FOS
         GOTO
                    TRANPRES
ENDLY
         ANOP
         GEN
                    3/3,2/0,1/0,10/%FOS
TRANPRES EXITM
         ENDM
YFIN
         DEFM
                    POS, TRANS
         BOUND
                    1H
         IFT
                    C'%TRANS' .EQ. C'LC', ENDLC
         GEN
                    3/4,2/2,1/0,10/%FOS
         GOTO
                    TRANPRES
ENDLC
         IFT
                    C'XTRANS' .EQ. C'LV', ENDLV
         GEN
                    3/4,2/1,1/0,10/%POS
         GOTO
                    TRANFRES
ENDLY
         ANOF
         GEN
                    3/4,2/0,1/0,10/%FOS
TRANPRES EXITM
         ENDM
```

LETTER	DEFM	CHAR, DELAY
has has 1 1 has 15	BOUND	1H
%FLAG	SET	0
701 ILTIC	IFP	C'%DELAY', NODELAY
%FLAG	SET	1
NODELAY	ANOP	
TTO E. I. I.	IFT	%FLAG .EQ. 1, ENDDELAY
	GEN	3/5,3/0,2/%DELAY
	DATAB	C'%CHAR'
	EXITM	W /MOTORIN
ENDDELAY	ANOF	
	GEN	3/5,3/0,2/0,8/C'%CHAR'
	ENDM	
COLOR	DEFM	COL
	BOUND	1H
	IFT	C'%COL' .EQ. C'RED', ORANGE
	GEN	3/6,10/0,1/1,2/0
ORANGE	IFT	C'%COL' .EQ. C'ORANGE', YELLOW
	GEN	3/6,10/0,1/1,2/1
YELLOW	IFT	C'%COL' .EQ. C'YELLOW', GREEN
	GEN	3/6,10/0,1/1,2/2
GREEN	IFT	C'%COL' .EQ. C'GREEN', NOPE
	GEN	3/6,10/0,1/1,2/3
NOPE	EXITM	
	ENDM	
TRIGGER	DEFM	TR
	BOUND	1H
	IFT	C'ZTR' .EQ. C'EXT', INT
	GEN	3/6,7/0,1/1,1/0,4/0
INT	IFT	C'%TR' .EQ. C'INT', END
	GEN	3/6,7/0,1/1,1/1,4/0
END	EXITM	
	ENDM	
CURSER	DEFM	
	BOUND	1H
	GEN	3/6,1/1,3/0,1/1,8/0
	ENDM	
LASTVECT		
	BOUND	1H
	GEN	3/6,1/0,1/1,11/0
	ENDM	

```
CHAREX
         EQU
         LW
                   R6.TCW
                                   LOAD TCW
                   R6 . X'120'
         STW
                                   STORE AT TOW LOCATION
REFRESH
         TD
                   DISPUNIT, X'8000'
                                        TEST DISPLAY UNIT
                                   TEST AGAIN IF DELAY
         BCT
                   1,$-1W
         BCT
                   2,$-2W
                                   TEST AGAIN IF CHANNEL ACTIVE
         CD
                   DISPUNIT, X'A000'
                                        DISPLAY CHARACTERS
                                   LOAD WAIT COUNT
         LI
                   R6,4000
WAIT
         SUI
                                   DECREMENT WAIT COUNT
                   R6,1
         CI
                   R6,0
                                   SEE IF ZERO YET
         BCT
                   GT, WAIT
                                   BRANCH BACK IF NOT YET ZERO
                                   BRANCH BACK TO REFRESH SCREEN
         BU
                   REFRESH
         BOUND
                   1 W
                                   ALIGN TOW
TCW
         EQU
                                   TRANSFER CONTROL WORD
         GEN
                   12/20,20/H(DATA)
                                        20 HALFWORDS OF DATA, ADDRESS
                                   OF DATA TO BE TRANSFERED
DATA
         EQU
         XINIT
                   100
                                   SET X INITIAL POSITION
                   100
                                   SET Y INITIAL POSITION
         YINIT
                                   DISPLAY 'A' AT FOSITION (100,100)
         LETTER
                   A
                   900
                                   CHANGE X POSITION TO 900
         TINIX
                                   DISPLAY 'B' AT POSITION (900,100)
         LETTER
                   B
                                   CHANGE Y CO-ORDINATE TO 900
         YINIT
                   900
                                   DISPLAY 'C' AT POSITION (900,900)
         LETTER
                   C
         TINIX
                   100
                                   CHANGE X CO-ORDINATE TO 100
         LETTER
                   D
                                   DISPLAY 'D' AT POSITION (100,900)
         XINIT
                   500
                                   CHANGE X CO-ORDINATE TO 500
                                   CHANGE Y CO-ORDINATE TO 500
                   500
         YINIT
                                   DISPLAY 'E' AT POSITION (500,500)
         LETTER
                   E
                   100,LC
         XINIT
                                   CHANGE X AND DISPLAY LAST CHAR
         XINIT
                   900, LC
                                   CHANGE X AND DISPLAY LAST CHAR
                                   NOTICE THAT MAXIMUM DELAY IS
                                   ASSUMED BY THE MACRO 'LETTER'
                                   THIS IS NEEDED BECAUSE OF THE
                                   DISTANCE BETWEEN CHARACTERS
         SCREEN SHOULD LOOK LIKE THIS
******************
         D
                                   C
         E
                     E
                                   E
```

\*

CHAREX

END

The display unit will draw a vector upon the reception of three types of data: 1) a "y final" data halfword, 2) any type of data halfword that specifies an x or y co-ordinate and has the "vector last transfer" bit set, and 3) a data halfword with an identifier code of 6 and the "vector last transfer" bit set.

manded to do so. However, to synchronize the display unit with the radar, a feature of the display unit causes the vector not to be drawn until an external signal is received. Upon reception of the external signal, the display unit draws the vector while modulating its intensity according the the feedback from the radar beams. To accomplish this task, the programmer must simply transfer a data halfword to the display unit with an identifier code of 6 that has bit 10 set and has bit 11 reset. All vectors drawn thereafter will be synchronized and modulated until another halfword is transferred with an identifier code of 6 that has both bits 10 and 11 set. Example Program 2 illustrates the use of vectors.

#### 3. Color Changing.

In order to accommodate the variation of colors for certain characters or vectors, the display unit allows the programmer to select the color (red, orange, yellow, or green) that data is to appear on the screen. To allow color changes, bit 13 of data halfwords with identifier codes of 6 is used to enable color changes. If bit 13 is set, then all data transferred after this halfword will appear as

```
PROGRAM V CTOR
                        TONY JACKSON
          DESCRIPTION: EXAMPLE PROGRAM FOR
                         COLOR DISPLAY UNIT
 DISPUNIT BQU
                  x'40'
                                              DISPLAY UNIT
    (REGISTER EQUATES AND MACROS AS IN EX.1)
V. CTOR
          EQU
                   A6, TCW
          LW
                                              LOAD TOW
                   R6, X'120'
          STW
                                             STORE AT TOW LOCATION
REFRESH
          TD
                   DISPUNIT, X'8000'
                                         TEST DISPLAY UNIT
TEST AGAIN IF DELAY
          BCT
                   1,3-1W
          BCT
                   2,$-2W
                                          TEST AGAIN IF CHANNEL ACTIVE
DISPLAY VECTORS
          CD
                  DISPUNIT, X'A000'
          LI
                   H6,4000
                                             LOAD WAIT COUNT
                   R6, 1
WAIT
          SUI
                                             DECREMENT COUNT
                   R6, 0
          CI
                                              COMPARE TO ZERO
          BCT
                   GT, WAIT
                                             IF NOT ZERO, WAIT
          BU
                   REFRESH
                                             IF ZERO, REFRESH
          BOUND
                   1W
TCW
          EQU
                                             TRANSFER CONTROL WORD
          GN
                   12/5,20/H(DATA)
                                             5 HALFWORDS OF DATA
DATA
          F.O.A
          XINIT
                   100
                                             INITIAL X
          YINIT
                   100
                                             INITIAL Y
          XFIN
                   900
                                             FINAL A
          YFIN
                   900
                                             FINAL Y
         YFIN
                   100
                                             NEW FINAL Y
         DRAWS TWO VECTORS AT A 45 DEGREE ANGLE
```

END

VECTOR

the color designated in bits 14-15 of this halfword.

Due to the restrictions on the color switching rate, it is recommended that all data of one color be transferred before changing colors to keep from changing too fast. Example program 3 illustrates color changing.

# C. Input Programming.

To input data from the display unit, there are two methods: keyboard input and trackball input.

# 1. Keyboard input.

In order to allow the operator to communicate with the CPU, the display unit allows keyboard input to the program. This is accomplished by the excution of CD with bit 25 on. It is important to note that only one character is transferred for each CD executed. Thus, if more than one character is to be transferred, one CD must be executed for each character. The ASCII character code is contained in the last eight bits of the halfword of data transferred. Condider the example program 4 that reads the keyboard.

#### 2. Trackball input.

If the position of the cursor is desired by the programmer, he simply issues a CD with bits 23 and 24 set. Upon execution of this instruction, two halfwords of data will be transferred to the address specified by the TCW, consecutively in core. These halfwords will contain the x and y co-ordinates of the cursor, respectively, in bit 6-15 of each halfword of data. Example program 5 illustrates trackball input.

## PROGRAM CHANGE

```
TONY JACKSON
         AUTHOR:
         DESCRIPTION: EXAMPLE PROGRAM FOR
                      COLOR DISPLAY UNIT
                X.40.
DISPUNIT DQU
    (REGISTER EQUATES AND MACROS AS IN EX. 1)
CHANGE
         Bau
                 R6, TCW
         LW
                                         LOAD TOW
                 R6, X'120'
         STW
                                         STORE AT TOW LOCATION
                 Re DCLR
                                         OBTAIN COLOR CONTROL
REFRESH
         LH
         STH
                 CLR
                                         PUT IN LIST
                 DISPUNIT, X'8000'
                                       TEST DISPLAY UNIT
         TD
         BCT
                                         TEST AGAIN IF DELAY
                 1.5-1W
         BCT
                                         TEST AGAIN IF CHANNEL ACTIVE
                  2,$-2W
                  DISPUNIT, X'A000' DISPLAY VECTORS
         CD
                 R6,4000
                                        LOAD WAIT COUNT
         LI
WAITL
         SUI
                 R6,1
                                         DECREMENT
         CI
                                       COMPARE TO ZERO
                 R6,0
         BCT
                 GT, WAIT1
                                         WAIT IF NOT ZERO
                 GRNCLR
                                       OBTAIN COLOR CONTROL
         LH
         STH
                 CLR
                                         PUT IN LIST
                 DISPUNIT, X'8000'
1,$-1W
2.$-2W
         TD
                                         TEST DISPLAY UNIT
         BCT
                                       TEST AGAIN IF DELAY
                                      TEST AGAIN IF CHENNEL ACTIVE
                  2,5-2W
         BCT
                 DISPUNIT, X'A000'
         CD
                                         DISPLAY VECTORS
         Ll
                 R6 4000
                                         LOAD WAIT COUNT
WAIT2
         SUI
                 R61
                                         DECREMENT
         CI
                 R6.0
                                         COMPARE TO ZERO
         BCT
                 GT.WAIT2
                                         WAIT IF NOT ZERO
         BU
                 REFRESH
                                         REFRESH SCREEN
         BOUND
                 1W
                 12/6,20/H(DATA) TRANSFER CONTROL WORD
TCW
         GAN
DATA
         EQU
                 c. .
CLR
         DATAH
                                          COLOR OF VACTORS
         XINIT
                 100
                                          INITIAL X
         YINIT
                 100
                                         INITIAL Y
                 900
         XFIN
                                         FINAL X
         YFIN
                 900
                                         FINAL Y
                 100
         YFIN
                                          NEW FINAL Y
         DISPLAYS TWO VECTORS AT A 45 DEGREE ANGLE
         AND ALTERNATES THE COLORS BETWEEN RED
         AND GREEN
REDCLR
         COLOR
                 RED
GRNCLR
         COLOR
                 GREEN
         END
                 CHANGE
```

PROGRAM KBD

```
AUTHOR:
                         TONY JACKSON
          DESCRIPTION: EXAMPLE PROGRAM FOR
                         COLOR DISPLAY UNIT
                   X . 40 .
DISPUNIT MAD
    (REGISTER EQUATES AND MACROS AS IN EXAMPLE 1)
KBD
          QU
          LW
                    R6, RD TCW
                                               ROAD KBD TCW
          STW
                    R6, X'120'
                                               STORE AT TOW LOCATION
          TD
                    DISPUNIT, X'8000'
                                               TEST DISPLAY UNIT
          BCT
                    1,5-1
                                               TAST AGAIN IF DELAY
                    2,3-2.
          BCT
                                              TEST AGAIN IF CHANNEL ACTIVE
                    DISPUNIT, X'B040'
          CD
                                              READ KBD CD
          TD
                    DISPUNIT, X'8000'
                                              TEST DISPLAY UNIT
          BCT
                   1,5-1W
                                              TEST AGAIN IF DELAY
                    2, 3-21
          BCT
                                              TEST AGAIN IF CHANNEL ACTIVE
          LB
                    KBDIN+1B
                                               CHARACTER RECEIVED
          STB
                    CHAR+1B
                                              PUT IN LAST BYTE OF
                                              CONTROL WORD
          LW
                    R6, DISPTC
                                              DISPLAY TOW
          STW
                    R6, X'120'
                                              STORE AT TOW LOCATION
                    DISPUNIT, X'A000'
          CD
                                              DISPLAY CHARACTER
                    R6, 4000
R6, 1
          LI
                                              LOAD WAIT COUNT
MAIT
          SUI
                                              DECREMENT
          CI
                    R6, 0
                                              COMPARE TO ZERO
          BCT
                   GT, WAIT
                                              IF NOT ZERO, WAIT
          BU
                   KBD
                                              NEXT CHARACTER
          BOUND
                   1%
                                              NEXT CHARACTER
RDTCH
          EQU 
          GLIN
                   12/1,20/H(KBDIN)
                                              RLAD KBD TCW
DISPICH
          Ugu
          GEN
                   12/3,20/H(DISP)
                                              DISPLAY TOW
KEDIN
          EQU
                   C.
          DATAH
          DATAH
                   C.
DISP
          UEH
                   500
          XINIT
                   500
          YINIT
          USiL
                   $
          LOTTER
                   X
          END
                   KBD
```

## D. Interrupt Processing.

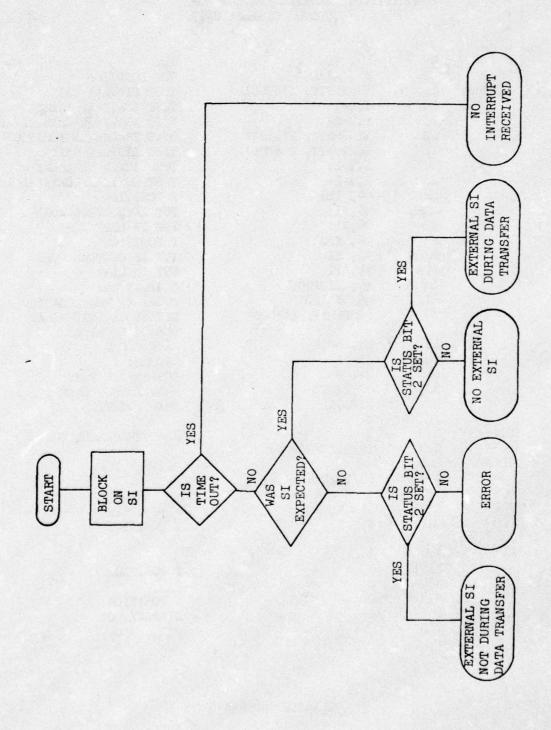
In order for the programmer to determine when the "trackball enter" button has been depressed or when data is to be entered from the keyboard, an interrupt is generated by the display unit. Actually what this amounts to is that bit 2 of the status bits is set and the fact that an interrupt is outstanding is noted in the semaphore for the CRT.

The programmer must check this semaphore occasionally if the input features of the display unit are to be utilized during normal processing. In order to check the semaphore the programmer simply does a CALM instruction with an operand field of BLOCK. Register 6 must contain the C-LIST index for the serial data terminal which is 168. Register 7 must contain the maximum amount of time that the program is to stay blocked in the event that in SI has occurred. Note that this time is in 1/60 sec. units.

There are two types of SI's that the programmer should be concerned with: 1) an external SI during the transfer of data to the display unit and 2) an external SI not during data transfer. These types should be distinguished because when data transferral is complete, an SI is generated regardless of whether the trackball position is to be read or the keyboard is to be read. To accommodate this characteristic, a flag is usually set to indicate that an SI is expected (done immediately before a CD instruction). The flowchart 1 illustrates the normal interrupt processing routine. Example program 6 illustrates the processing of interrupts during data transferral and not during data transferral.

```
PROGRAM CURS
          AUTHOR: TONY JACKSON
          DESCRIPTION: EXAMPLE PROGRAM FOR
                        COLOR DISPLAY UNIT
CURS
          BOU
                  R6, TCW
R6, X'120'
          LW
                                              TCW
                                           TCW LOCATION
TEST DISPLAY UNIT
          STW
                   DISPUNIT, X'8000'
NXTCUR
          TD
          BCT
                   1,3-1%
                                           TEST AGAIN IF DELAY
                                          TEST AGAIN IF CHANNEL ACTIVE READ TRACKBALL POSITION
          BCT
                   2,5-2W
                   DISPUNIT, X'B180'
          CD
          TD
                   DISPUNIT, X'8000'
                                             TEST DISPLAY UNIT
          BCT
                   1,$-1%
                                              TEST AGAIN IF DELAY
          BCT
                   2,$-2W
R6, TB1
                                             TEST AGAIN IF CHANNEL ACTIVE
          LH
                                            X POSITION
                   R6,X1
          ORMH
                                             PUT IN CONTROL WORD
                   R6,X1
          STH
                                             PUT IN LIST
                   R6, TB2
R6, Y1
R6, Y1
R6, DISPICA
          LH
                                            Y POSITION
                                         PUT IN CONTROL WORD
          ORMH
                                          PUT IN LIST
DISPLAY TOW
          STH
          L
                 R6, X'120'
          ST
                                          STORE AT TOW LOCATION DIPLAY CHARACTERS AT
                  DISPUNIT, X'8000'
         CD
                                            CURSOR LOCATION
                  R6, 4000
                                             WAIT COUNT
WAIT
        SUI
                  R6, 1
                                             DECREMENT
          CI
                   R6, 0
                                             COMPARA TO ZERO
          BCT
                  GT. WAIT
                                             WAIT IF NOT ZERO
          BU
                   NXTCUR
                                             NEXT POSTTION
          BOUND
                   1.
          EQU
TCM
                                              READ TRACKBALL TOW
                  12/2,20/H(POS)
         GAN
DISPTCH EQU
                                              DISPLAY TOW
      GAN 12/3,20/H(CHAR)
BQU $
DATAH C'
DATAH C'
POS
TBl
                                              X POSITION
TB2
                                              Y POSITION
        ±QU
±QU
CHAR
Xl
          TIMIX
                  0
                                              X POSITION
Yl
         riQU
         YINIT O
                                              Y POSITION
         LETTER C
                                              DISPLAY "C"
```

END



```
PROGRAM SERV
          AUTHOR: TONY JACKSON
          DESCRIPTION: EXAMPLE PROGRAM FOR
                        COLOR DISPLAY
    (REGISTER EQUATES AND MACROS AS IN EX. 1)
                   CLIST
          EXT
                   X'40'
DISPUNIT EQU
SARV
          EQU
                   R6,4000
                                              WAIT COUNT.
          LI
                                              CAPABILITY INDEX.
          Lil
                   CLIST
                   .BLOCK
                                              WAIT FOR SI.
          CALM
          BCT
                   14,5±RV
                                              IF OUT OF TIME.
                                              POLL AGAIN.
          BCT
                   I-TXE.O
                                              IF SI GET OUT.
          UL
                   SIRV
                                              ELSE POLL AGAIN.
EXTSI
          EQU
                   3
T.ST1
          L
                   H6,TCW
                                             TCW
          STW
                   R6,X'20'
                                             TCW LOCATION
          TD
                   DISPUNIT, X'2000'
                                              TRANSFER STATUS BYTES
          ECT
                   ANY, TEST1
                                             ANY CONDITION, RETEST
                   O, MXPOCT
          7BM
                                              ZERO AND TEST EXPECTED
                                              BIT
          BCT
                   SAT, TRANS
                                              IF ON, DATA TRANSFER
          Ton
                   2,STAT
                                              TEST STATUS BIT
                   S.T, DATASI
                                              IF ON, GOOD SI
          (BRROR ROUTINE OF SOME SORT)
DATASI
          UCV
          (ROUTING TO TEST KBD AND TRACKBALL STATUS BITS
          AND PERFORM TASK. MUST BRANCH BACK TO "SERV"
          TO BLOCK AGAIN)
          BU
                   SERV
TRANS
          JU
                   3
          TSH
                   2,STAT
                   SST. DATASI
                                              IF ON GET DATA
          (ROUTING TO COMPLETE FUNCTIONS INITIATED IN "DATASI")
                   SERV
         NOT THAT DATA TRANSFER IS INITIATED IN "DATASI".
          CONTROL WILL RETURN TO "TRANS" TO COMPLETE THESE
          TASKS. "DATASI" WILL SET THE EXPECTED FLAG. DATAB X'00'
```

X'0000'

GEN 12/1,20/H(STAT)

SERV

DATAH

END

EXP. CT

STAT

## PART FIVE

# AN ADAPTIVE GRADIENT BASED TECHNIQUE FOR ACCELERATING THE CONVERGENCE TIME OF ADAPTIVE ARRAY ANTENNAS

E. R. Graf and H. T. Nagle, Jr. Co-Project Leaders

ELECTRICAL ENGINEERING
AUBURN UNIVERSITY

U. S. ARMY MISSILE COMMAND REDSTONE ARSENAL, ALABAMA

#### Foreword

This is part of the final technical report of a study being conducted by the Electrical Engineering Department under the auspices of the Engineering Experiment Station of Auburn University. This report is submitted toward partial fulfillment of Contract DAAHO1-71-C-1303, task 2.1.13, granted to the Engineering Experiment Station, Auburn, Alabama, by the U.S. Army Missile Command, Redstone Arsenal, Alabama.

## AN ADAPTIVE GRADIENT BASED TECHNIQUE FOR ACCELERATING THE CONVERGENCE TIME OF ADAPTIVE ARRAY ANTENNAS

D. M. Jones, D. G. Burks, and E. R. Graf

#### **ABSTRACT**

A significant improvement in the convergence characteristics of the control loops of the adaptive phased array antenna system proposed by Applebaum [1] is realized by incorporating a technique of adaptive filter initialization into the existing system. The fundamental strength of this technique lies in the fact that the convergence time of the resulting control system is limited primarily by electronic sampling considerations rather than by the natural modes or eigenvalues of a feedback control system. Noise bandwidth is the critical factor that determines how closely the initial weights predicted by this technique approach the theoretically optimum values.

A phase analysis of the first order control loops of Applebaum shows that in the monochromatic case the phases on the various lines in the control loops remain constant during the convergence period. This result suggests that the phase information for each steering weight permutation

may be obtained from a single sample of the instantaneous inputs to the integrators. It is further demonstrated that under these conditions the final output values of all of the control loop integrators are linearly related to their initial input values by the same real number, k. The proper k is that value for which d (array output)/dk = 0. k is determined by an adaptive output gradient procedure. Thus, in the monochromatic case, the optimum weight vector can be determined exactly by appropriately processing a single sample of the instantaneous inputs to the integrators. This fact is consistent with the earlier results of Brennan [19].

In the case of non-zero bandwidth, the antenna element voltages will vary. Consequently, the phases of the integrator inputs do not remain constant through the transient convergence period as they do in the zero bandwidth case. Therefore, in this case, a single sample of the initial phases of the integrator inputs does not provide sufficient information to accurately estimate the final values of the element weights. However, through repeated application of the adaptive initialization procedure, updated phase information is acquired, and the filters can be initialized such that the signal-to-noise ratio after the final iteration is to within approximately 3 dB of the optimum value. The system, initialized in this manner, completes the convergence process in the normal sense. Simulation indicates that initialization to within 3 dB of the optimum signal-to-noise ratio usually is effected in three or four iterations.

Principally, the noise bandwidth determines to what degree each iteration can improve the signal-to-noise ratio. Due to these considerations, application of this technique is limited to CW systems.

## TABLE OF CONTENTS

LIST	OF F	IGURES
ı.	INT	RODUCTION
II.	ADA	PTIVE PROBLEM FORMULATION
111.	CON	TROL LOOP ANALYSIS
	A)	Equivalent control loop
	В)	Control loop signal phase analysis and system modification - monochromatic case
IV.	ADA	PTIVE SYSTEM SIMULATION
	A)	Adaptive simulation incorporating the gradient procedure in the case of non-zero bandwidth
	B)	Broadband applications
٧.	CON	CLUSIONS
REFER	ENCE	S

# LIST OF FIGURES

1.	Linear phased array antenna with element spacing, d, steered to angle $\theta$
2.	Adaptive phased array representation showing the matrix transformation that decorrelates all correlated antenna element noise, followed by the complex combiner, W
3.	Block diagram of a Howells-Applebaum first order adaptive control loop used for coherent noise suppression
4.	Complex element weight trajectory during the convergence period for a 12 element adaptive array steered to 135 degrees in the presence of three coherent noise sources at 70, 85, and 100 degrees from endfire. Zero bandwidth case
5.	Initial antenna pattern of a 12 element adaptive array steered to 135 degrees with 3 coherent noise jammers at 70, 85 and 100 degrees from endfire. Zero bandwidth case
6.	Complex integrator output trajectory corresponding to the element weight of Figure 4 during the convergence period
7.	Complex array output trajectory during the convergence period for a 12 element adaptive array steered to 135 degrees in the presence of 3 coherent noise sources at 70, 85 and 100 degrees from endfire. Zero bandwidth case
8.	Optimal pattern of a 12 element adaptive array steered to 135 degrees in the presence of 3 coherent noise sources at 70, 85, and 100 degrees from endfire. Zero bandwidth case
9.	Adaptive control loop representation equivalent to the Howells-Applebaum first order control loop

10.	Superposition of retrodirective coherent noise cancellation pattern on the unadapted steered array pattern				31
11.	Block diagram of an adaptive control loop incorporating the adaptive initialization modification. Zero bandwidth case	•	•		36
12.	Block diagram of a general adaptive control loop incorporating adaptive initialization for narrow bandwidth coherent noise cancellation		•	•	38
13.	Improvement in signal-to-noise ratio after the first iteration of the adaptive initialization procedure as a function of coherent noise bandwidth. Optimum improvement after convergence - 31 dB			•	41
14.	Comparison of convergence time of adaptive array with a) no adaptive initialization b) one iteration of adaptive initialization c) two iterations of adaptive initialization d) three iterations of adaptive initialization				42

### I. INTRODUCTION

The theory of phased array antenna systems is highly developed with respect to both beam shaping and electronic steering. From this basis the theory of adaptive arrays has evolved. An adaptive array utilizes a control system sensitive to the environment to configure the amplitude and phase of each element weight individually so as to maximize some predetermined figure of merit selected according to the nature of the application.

Applebaum [1] provided the initial significant treatment of the adaptive problem. In that work he determined a unique element weight vector which maximizes a signal to noise ratio defined as the ratio of signal power output to total noise power output. This ratio is the appropriate figure of merit for radar applications. Widrow [2] expanded the theory to include adaptive systems which maximize the expected value of a received signal. Brennan et. al. [3, 5] have contributed greatly to the theory of adaptive control loop noise analysis. This author concentrates on the convergence rate problem of adaptive arrays used in radar applications.

In Applebaum's work the equation for the adapted weight vector is expressed as an optimal control law, and a first order control system is presented which, in steady state, provides a good approximation to this law. Subsequent research by Brennan [3, 5] has shown that given certain spatial noise configurations, the convergence time for the first

order control loops may often exceed  $16\mu s$ . In fact, during the course of the investigation reported herein, several practical examples were simulated in which convergence required in excess of  $80\mu s$ , which is far greater than the allowable time for many radar applications (as low as  $4\mu s$ ).

Brennan further demonstrated that the convergence rate of this control system is proportional to the eigenvalues of the noise covariance matrix generated from the quiescent antenna element noise voltages. Brennan [3] and Widrow [2] both have shown that whenever the eigenvalues of the noise covariance matrix vary widely, a poor convergence rate results. In order to accelerate the convergence process, Brennan proposed a second order control system in which the resulting convergence rate is increased by a factor of almost two. In this case the two time constants of convergence are still related (but not proportional) to the eigenvalues of the covariance matrix. Brennan concludes that this system's response is still not adequate for radar purposes.

Reed et. al. [19] have formulated a digital processing scheme which insures covergence of the signal-to-noise ratio to within 3 db of optimum by taking approximately 2N independent samples of the input process, forming a sample covariance matrix and inverting that matrix to form an approximation to the well known optimal control law,

Wopt = M<sup>-1</sup>S\*, where Wopt is the optimum weight vector, M<sup>-1</sup> is the inverse of the noise covariance matrix, and S\* is the vector of steering weights applied to the array. Undoubtedly, this method would be an excellent solution to the convergence problem were the electronics to implement the scheme available.

In this study a modification of Applebaum's original circuit is developed in which a near optimum weight vector is generated in 4 or less iterations of an adaptive initialization procedure developed herein. Simulation indicates that application of this technique generally yields a set of weights which result in the signal-to-noise ratio of the array output being within 3 to 4 dB of optimum. Once initialized, the array proceeds to converge to the final optimum value in the conventional manner with, consequently, a significant factor of improvement in the overall convergence time to within 3 dB of optimum. Quite often, in fact, the initialization procedure yields a signal-to-noise ratio that is already within 3 dB of optimum.

In the following section a brief statement of the original problem will serve to define the notation, and a summation of Applebaum's results will provide the departure point for this work.

## II. ADAPTIVE PROBLEM FORMULATION

Consider the n element array of Figure 1. When a signal is incident from the  $\theta$  direction, the voltages in the antenna elements,  $\mathbf{s}_i$ , are proportional to

$$s_{i} = e^{-j i kd \cos \theta}$$
 (1)

where

 $k = 2\pi/\lambda$ ,  $\lambda = wavelength$ 

d = element spacing

 $\theta$  = angle of incidence measured from endfire

i = element number

The signal reference point, P, is a distance d from the first antenna element along the extended array axis as shown in Figure 1.

If the antenna is steered in the  $\Theta$  direction, the conventional element weights are proportional to

$$w_i = e^{+j} i kd \cos \theta$$

$$w_i = s_i^*$$
(2)

where \* denotes the conjugate operator.

These weights are conveniently expressed in vector notation as

$$W = S^* = \begin{bmatrix} s_1^* \\ s_2^* \\ \vdots \\ s_n^* \end{bmatrix}$$
 (3)

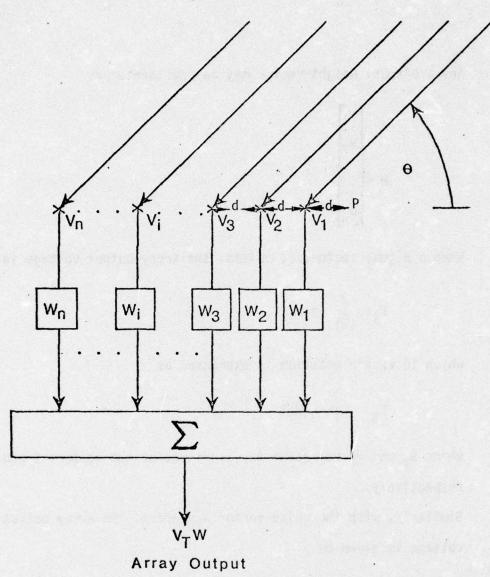


Figure 1. Linear phased array antenna with element spacing, d, steered to angle 0.

Any arbitrary weight vector may be represented as

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$
 (4)

When a signal vector, S, exists, the array output voltage is

$$E_{S} = \sum_{i=1}^{n} s_{i} w_{i}$$
 (5)

which in vector notation is expressed as

$$E_{S} = S_{T}W = W_{T}S \tag{6}$$

where  $S_T$  and  $W_\tau$  represent the transpose of the vectors S and W, respectively.

Similarly, with the noise vector N present, the array output noise voltage is given by

$$E_{N} = N_{T}W = W_{T}N \tag{7}$$

Another parameter of interest is the noise covariance matrix, M, defined as the array of elements

$$m_{i,j} = E \left[ n_i^* \ n_j \right] \tag{8}$$

where  $n_i$  = noise voltage on the i<sup>th</sup> element and E denotes the expectation operator. Note that  $m_{ij}$  is proportional to the noise power in the i<sup>th</sup> element and that  $m_{ij}$  is the covariance of the i<sup>th</sup> and j<sup>th</sup> noise voltages.

The covariance matrix, M, can, in general, be expressed as the sum of two independent matrices, one a covariance matrix of correlated noise sources (external jammers) and the second, a covariance matrix of uncorrelated noise sources (in-channel component noise and random sky noise). The second matrix by nature is a diagonal matrix.

$$M = E[N*N_T] = E[(N_C)*(N_C)_T] + M_N$$
 (9)

N = column vector of the total noise voltages

 $N_c$  = column vector of correlated noise voltages

M<sub>N</sub> = diagonal noise covariance matrix from uncorrelated sources

The noise covariance matrix, M, is a positive definite, Hermitian matrix. Consequently a unitary transformation, X, exists that will diagonalize M.

$$X^{-1} M X = \Lambda$$
 (10)

where  $\Lambda$  is the diagonal matrix whose elements are the eigenvalues of M. There further exists a real diagonal matrix B of rank N which will normalize  $\Lambda$  to the identity matrix, I. This transformation is of the form

$$B^* \wedge B_T = I \tag{11}$$

or

$$B* X^{-1} M X B_T = I$$
 (12)

This representation can be further simplified by using the unitary property

$$X_{\mathsf{T}}^{\star} = X^{-1} \tag{13}$$

Making the assignment  $C^* = \chi^{-1}$ , then

$$X = (X^{-1})^{-1} = (X^{-1})_{T}^{*} = (C^{*})_{T}^{*}$$
 (14)

Thus,

$$X = C_{\mathsf{T}} \tag{15}$$

and

$$x^{-1} = c*$$
 (16)

Using these properties, equation (12) can be written as

$$B* C* M C_T B_T = I$$
 (17)

or since

$$M = N * N_{T}$$
 (18)

$$B^* C^* N^* N_T C_T B_T = I$$
 (19)

which becomes

$$(BCN)^* (BCN)_T = I$$
 (20)

Letting D = BC

$$(DN)^* (DN)_T = I \tag{21}$$

D is the matrix that will decorrelate and equalize the noise power in each of the array element channels. This transformation is represented in Figure 2, followed by a combiner with the set of weights,  $\hat{W}$ .

Thus, any correlated noise appearing at the array aperture will be transformed to the uncorrelated noise vector,  $\hat{N}$ , as depicted in the figure.

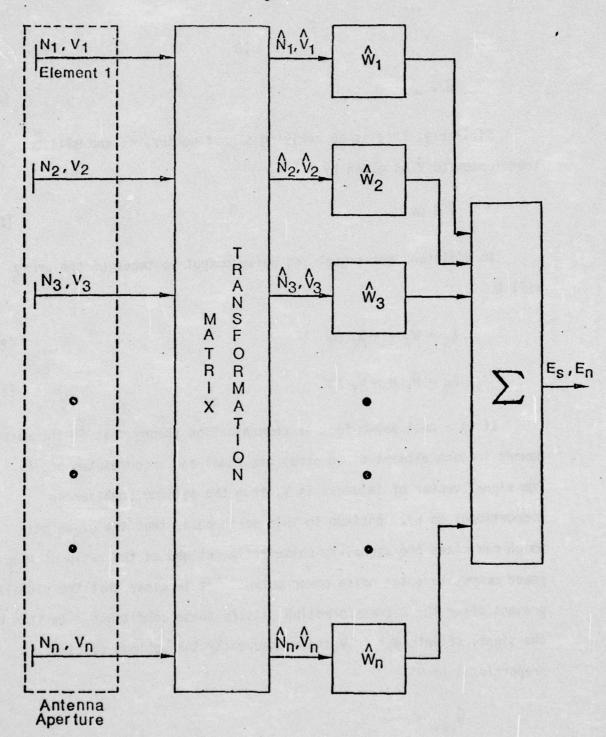


Figure 2. Adaptive phased array representation showing the matrix transformation that decorrelates all correlated antenna noise, followed by the complex combiner,  $\hat{W}$ .

$$\hat{N} = DN \tag{22}$$

Similarly, if V is an arriving signal vector, it too will be transformed to  $\hat{V}$  as given by

$$\hat{\mathbf{V}} = \mathbf{D}\mathbf{V} \tag{23}$$

In addition, the signal and noise output voltages of the array will be

$$E_{S} = \hat{W}_{T} \hat{V} = \hat{W}_{T} DV \tag{24}$$

$$E_{N} = \hat{W}_{T} \hat{N} = \hat{W}_{T} DN \qquad (25)$$

It is a well known fact in communication theory that if the noise powers in each element of an array are equal and uncorrelated and if the signal vector of interest is V, then the optimum combiner is proportional to V\*. Optimum in this sense means that set of weights which maximizes the signal-to-noise ratio defined as the ratio of signal power output to total noise power output. It is clear that the signals present after the D transformation satisfy these conditions. In this case the signal of interest is  $\hat{V}$  and consequently the optimum combiner is proportional to  $\hat{V}^*$ 

$$\hat{W}_{\text{opt}} = \rho \hat{V}^*$$

$$= \rho D^* V^*$$
(26)

where p is the constant of proportionality.

By comparing equations (6) and (7) with (24) and (25), it is clear that an array containing the transformation may be implemented by the conventional phased array of Figure 1, if the weights in the conventional array are given by

$$W = D_{T} \hat{W} . \tag{27}$$

Thus, using equation (26), the optimum combiner is

$$W_{\text{opt}} = \rho D_{\text{T}} D^* V^* \tag{28}$$

However, from equation (21),

$$D^* N^* N_T D_T = I$$
 (29)

$$N^* N_T = D^{*-1} D_T^{-1}$$
, (30)

and since  $M = N*N_T$ , then

$$M = (D_T D^*)^{-1}$$
 (31)

Using this result, equation (28) becomes

$$W_{\rm opt} = \rho M^{-1} V^*$$
 (32)

From a slightly different point of view, the weight vector, W, that satisfies the control equation  $MW = \rho V^*$  provides the optimum set of weights under the noise condition described by M. This is the control law first formulated by Applebaum [1].

The system suggested by Applebaum for implementing the optimal control law is depicted in Figure 3. Note that each of the lines,  $z_i$ ,

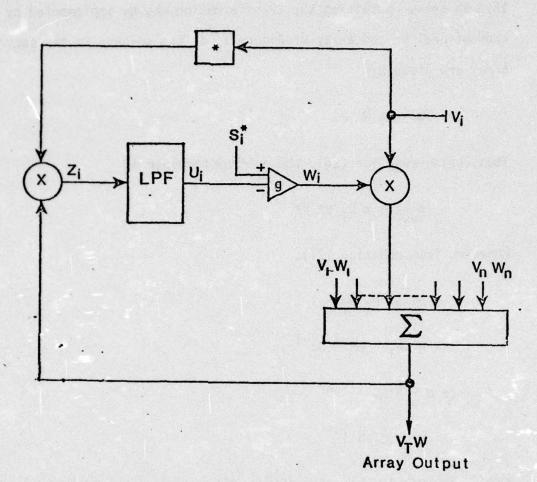


Figure 3. Block diagram of a Howells-Applebaum first order adaptive control loop used for coherent noise suppression.

constitutes an element of the vector that is the product of the covariance matrix, M, and the weight vector, W. Analysis of this system is performed as follows.

The complex weight vector, W, is given by

$$W = (S^* - U)g \tag{33}$$

where U is the output vector of the low pass filters, g is the gain of the summing amplifiers, and S\* is the steering vector. The state equation for the low pass filters is given by

$$\tau \dot{U} + U = Z \tag{34}$$

where  $\tau$  is the time constant of the filters and Z is the input vector to the filters. U denotes differentiation of U with respect to time.

The input to the low pass filters, Z, is

$$Z = V* (V_T W)$$
 (35)

[Note that  $V_T$  W is the system output.]

Since U is the output vector from the integrators, average and instantaneous values for U and W, respectively, are identical. The average value of Z, however, is

$$Z = \overline{V^* (V_T W)}$$
 (36)

$$= \overline{V} + \overline{V}_{T} W \tag{37}$$

It is important for later considerations to note that the equation for the average value of the integrator input, Z, is not the same as that for the instantaneous value of Z.

Since the steering vector, S\*, is constant.

$$\dot{U} = -\frac{\dot{W}}{g} \tag{39}$$

Equations (33) through (39) are solved simultaneously to yield

$$\frac{\tau}{g}W + (M + \frac{1}{g})W = \rho S^*$$
 (40)

where I is the identity matrix. In steady-state W = 0 and the system satisfies  $(M + \frac{I}{g})$  W =  $\rho$  S\*. If g is sufficiently large this equation becomes a good approximation to the optimal control law MW =  $\rho$  S\*. Equation (40) represents a system of N coupled differential equations. It is noted that the weight vector, W, that satisfies this system is a function of both the loop parameters and the external environment. A more complete discussion of this control loop is given in [1, 3, 5, 10]. It is sufficient here to point out several system constraints.

Brennan [3, 10] demonstrated that the element weights converge to their optimal values in an exponential manner when examined in a transformed space defined as follows. Since M is a Hermitian matrix of rank n, there exists a unitary transformation, X, which diagonalizes the covariance matrix.

$$x^{-1} M X = \Lambda \tag{41}$$

where  $\Lambda$  is the diagonal matrix whose non-zero elements are the eigenvalues of M. In this transformed space equation (40) becomes

$$\frac{\tau}{g} \omega + (\Lambda + \frac{I}{g}) \omega = \rho S^*$$
 (42)

where

$$\omega = XW \tag{43}$$

$$S = XS$$
 (44)

Since  $\Lambda$  is a diagonal matrix, equation (42) represents a set of N linear independent differential equations. Each  $\omega_n$  has a solution

$$\omega_{n} = \left\{ \omega_{ni} - \frac{\delta_{n}^{*}}{\lambda_{n} + \frac{1}{g}} \right\} \quad e^{-\left(\frac{1 + g\lambda_{n}}{\tau}\right)} \quad t + \frac{\delta_{n}^{*}}{\lambda_{n} + \frac{1}{g}}$$
(45)

where  $\omega_{ni}$  are the initial values of  $\omega_n$ . These weights converge to their final values with time constants  $\frac{\tau}{1+g\lambda_n}$ . Thus, it is evident that loop convergence characteristics are heavily dependent on both the characteristics of the control loop (g and  $\tau$ ) and on the environment (M).

Apparently, selection of parameters that yield a large  $\frac{g}{\tau}$  ratio would effect rapid convergence. Brennan, however, has demonstrated that this choice is not an effective solution. It is shown in [3] that the total noise power in the array output is given by

$$P_{n} = \overline{W}_{T}^{*} M \overline{W} \left[ 1 + \frac{g}{2\tau} \sum_{i=1}^{n} \lambda_{i} \right]$$
 (46)

where  $\overline{\mathbb{N}}$  is the vector of average weights to which the loops converge in the presence of no loop noise. It is clear that large values of  $\frac{g}{\tau}$  lead

to increased output noise power, and, consequently, to system instabilities. Under the conditions of widely varying covariance matrix eigenvalues and a large value of g, the system can become unstable. This consideration places a practical limit on the value of g. Simulation shows that for  $\tau=100\mu S$ , a value of g less than or equal to 10 yields good system stability. Examples of instability with g=15 have been observed during this study. Brennan concludes that when the eigenvalues of the covariance matrix are widely varied, no  $g/\tau$  ratio exists that will result in rapid convergence of the weights and simultaneously maintain a reasonable value of loop noise. A simulation of this control system under the zero bandwidth constraint has been performed and an example is given to demonstrate the convergence properties discussed.

The trajectory of an element weight from a 12 element linear array is shown in Figure 4. The array was steered to an angle of 135 degrees in the presence of three noise sources located at 70, 85 and 100 degrees from endfire, respectively, as depicted in Figure 5. The direction of the trajectory is as indicated. The time at certain points along the trajectory is also shown. The array requires approximately  $10\mu S$  to converge. Using the same format, the output of the integrator associated with the loop of the element weight of Figure 4, and the output of the complete array are shown in Figures 6 and 7, respectively. The optimal pattern is shown in Figure 8. Note that the phases of the signals of Figures 6 and 7 appear to be constant. This is, in fact, not a coincidence. As will be demonstrated in the following section,

AD-AU32 121
AUBURN UNIV ALA ENGINEERING EXPERIMENT STATION
REAL-TIME PHASED ARRAY RADAR STUDIES. (U)
1976
E R GRAF, H T NAGLE

DAAH01-71-C-1303
NL

50°5
Abasiai

E ND
DATE
FILMED
DATE
FI

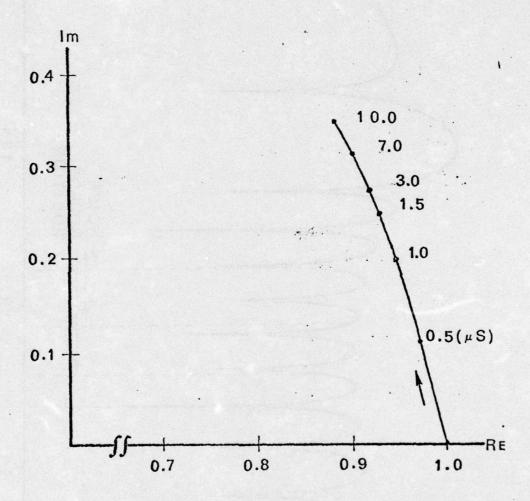
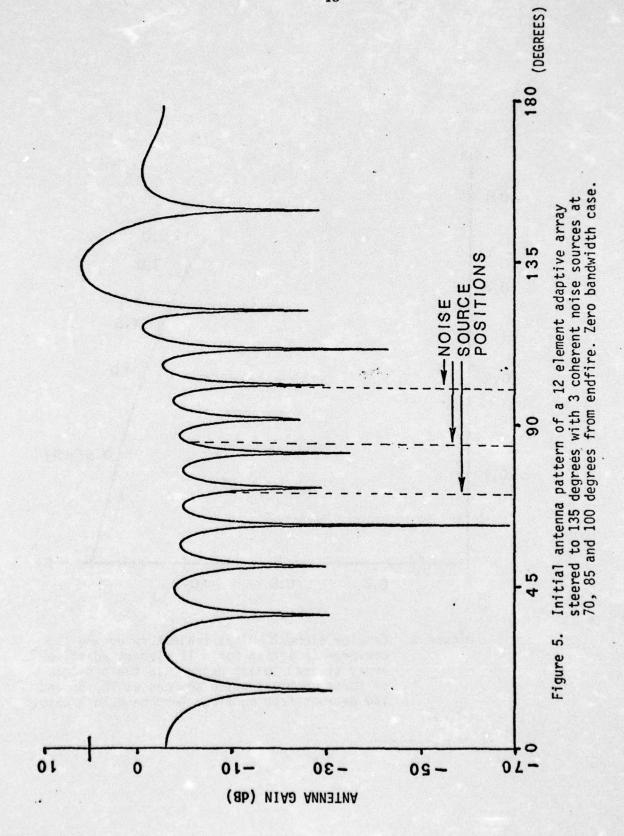


Figure 4. Complex element weight trajectory during the convergence period for a 12 element adaptive array steered to 135 degrees in the presence of three coherent noise sources at 70, 85 and 100 degrees from endfire. Zero bandwidth case.



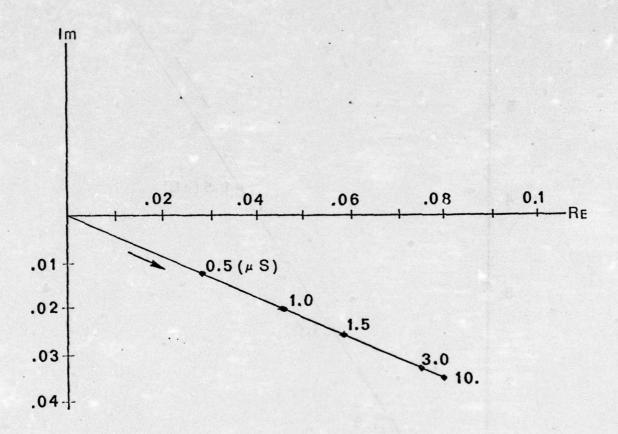


Figure 6. Complex integrator output trajectory corresponding to the element weight of Figure 4 during the convergence period.

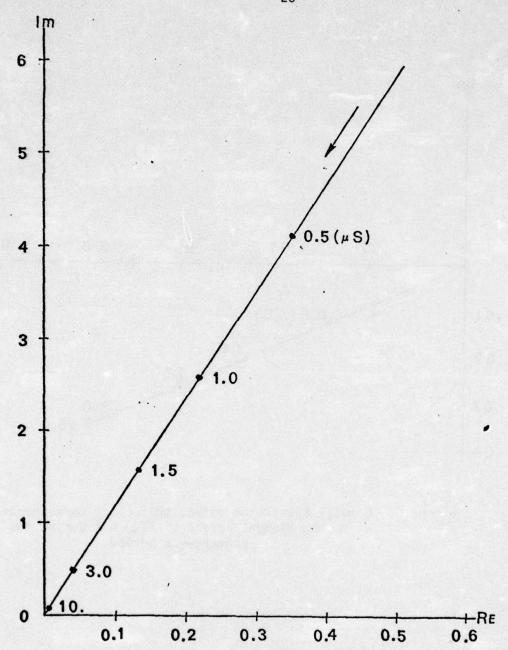
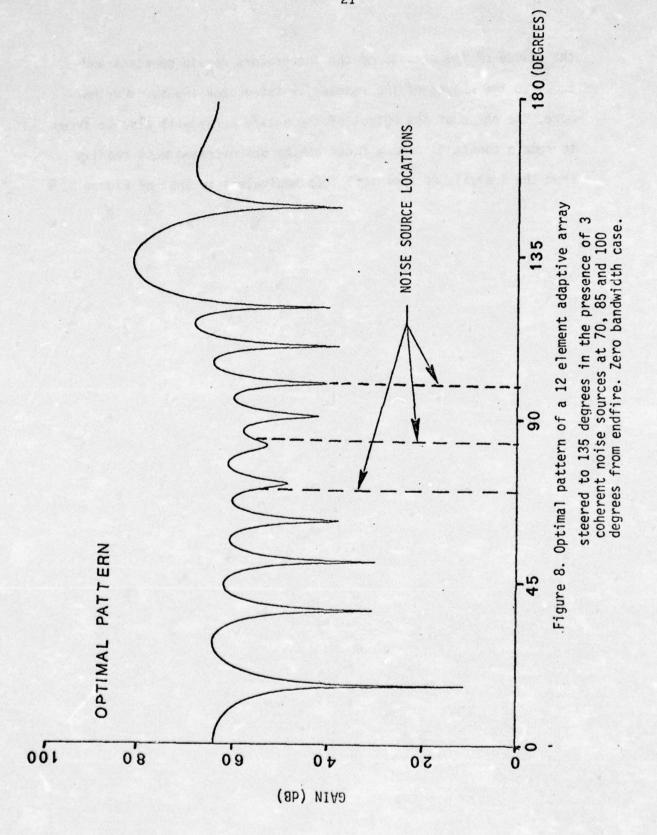


Figure 7. Complex array output trajectory during the convergence period for a 12 element adaptive array steered to 135 degrees in the presence of 3 coherent noise sources at 70, 85 and 100 degrees from endfire. Zero bandwidth case.



the phases of the outputs of the integrators remain constant and equal to the phases of the respective integrator inputs. Furthermore, the phase of the output of the entire array will also be shown to remain constant. These facts can be demonstrated most readily from the analysis of a control loop equivalent to that of Figure 3.

## III. CONTROL LOOP ANALYSIS

## A. Equivalent Control Loop

An adaptive control loop equivalent to that proposed by Applebaum [1] is shown in Figure 9. Equivalence of the two systems can be verified in the following manner. Let each circuit have the same input vectors  $S^*$  and V. First, assuming the two lines U and  $\hat{U}$  (from Figure 3 and 9 respectively) are equal, the outputs of each array are seen to be equivalent, since  $EI = gV_TS^*$  and  $E2 = -gV_TU$ . By closing the loops equivalence of U and  $\hat{U}$  is established. It is necessary, of course, that the low pass filters of both loops have identical parameters.

Section A of the equivalent system represents that division of the control loop which preserves the mainbeam and accounts for the basic pattern of the antenna. Its output, El, is just the output of a conventional array. The signal Fl is not generated in a conventional array, but is required for the formation of the adaptive pattern. As long as the phasor voltages induced at the antenna elements do not vary, the value of Fl remains constant. For this reason it is included in Section A.

Section B represents that part of the system which generates the adaptive or coherent noise reduction pattern. From this point of view the adaptive array may be viewed as a generalized form of a coherent sidelobe cancelling system. Consider, for example, the case in which the antenna is steered to the angle  $\theta_{\rm S}$ . In the quiescent environment the steering vector is

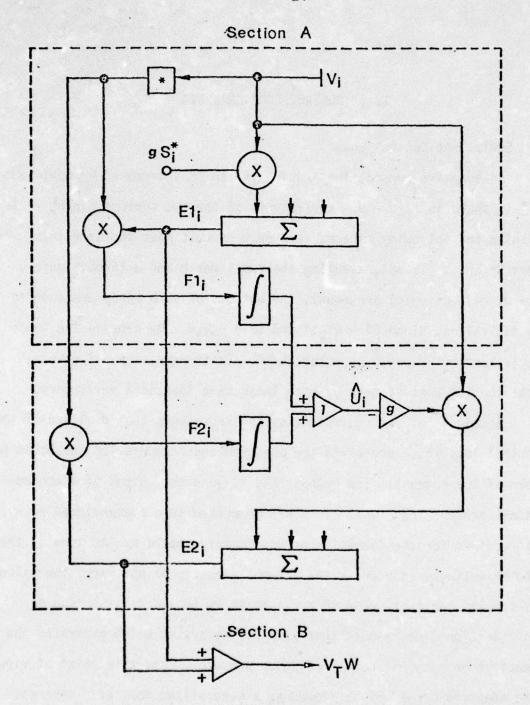


Figure 9. Adaptive control loop representation equivalent to the Howells-Applebaum first order control loop.

$$W_{q} = \begin{bmatrix} \alpha_{1} \\ +j\beta_{S} \\ \alpha_{2} \\ \vdots \\ +j(n-1)\beta_{S} \\ \alpha_{n}e \end{bmatrix}$$
(47)

where

$$\beta_{S} = \frac{2\pi}{\lambda} d \cos \theta_{S}$$
 (48)

and the  $\alpha_i$  are amplitude weights. Furthermore, under quiescent conditions the noise in each of the channels may be assumed equal and uncorrelated. Therefore, the quiescent covariance matrix would be the diagonal matrix

$$M_{\mathbf{q}} = \begin{bmatrix} p_{\mathbf{q}} & 0 & 0 & \cdots & \cdots & 0 \\ 0 & p_{\mathbf{q}} & & & 0 \\ \vdots & & & & \ddots \\ \vdots & & & & & \ddots \\ 0 & 0 & \cdots & & & p_{\mathbf{q}} \end{bmatrix}$$

$$(49)$$

where  $p_q$  equals the noise power output of each element. With the general amplitude weight,  $\alpha_i$ , the antenna pattern would be

$$G_{\mathbf{q}}(\beta) = \sum_{i=1}^{n} \alpha_{i} \exp[-j(i-1)(\beta-\beta_{s})]$$
 (50)

where

$$\beta = \frac{2\pi}{\lambda} d \cos \theta \tag{51}$$

In matrix notation this equation becomes,

$$G_{\mathbf{q}}(\mathbf{B}) = B_{\mathsf{T}} W_{\mathbf{q}} \tag{52}$$

where

$$B = \begin{bmatrix} 1 \\ exp[-j\beta] \\ exp[-2j\beta] \\ \vdots \\ exp[-j(n-1)\beta] \end{bmatrix}$$
(53)

The optimal weights under these conditions are, in fact, just  $\boldsymbol{W_{\!q}}$  , and

$$M_{q} W_{q} = p_{q} W_{q} (=_{P} V^{*})$$
 (54)

Thus, from equation 32, the control law for any noise environment is

$$MW = M_{\mathbf{q}} W_{\mathbf{q}} = p_{\mathbf{q}} W_{\mathbf{q}}$$

or

$$W = p_q M^{-1} W_q \tag{55}$$

Now consider the addition of a single coherent jammer located at  $\theta_J$ . The jamming signal (or coherent noise) in each element is then given by

$$n_{c}(i) = J(\omega) \exp[-j(i-1)\beta_{J}]$$
 (56)

where  $J(\boldsymbol{\omega})$  is the jamming signal in the first element and

$$\beta_{J} = \frac{2\pi}{\lambda} d \cos \theta_{J} \tag{57}$$

The elements of the covariance matrix of the jammer are

$$(M_j)_{k\ell} = p_j \exp[+j(k-\ell)\beta_j]$$
 (58)

where  $p_J$  is the envelope jamming power in each channel. Note that  $M_J$  is an  $n^{th}$  order Hermitian matrix in which all terms of each diagonal are identical. Furthermore, it may be factored and expressed as

where

$$H = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ 0 & e^{-j\beta_{ij}} & & & \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & e^{-j(n-1)\beta_{ij}} \end{bmatrix}, \qquad (59)$$

and Q is an nth order unity matrix.

As discussed in Section II, the expression for the covariance matrix in this environment (quiescent plus jammer) is the sum of the two independent covariance matrices  $M_{\rm q}$  and  $M_{\rm J}$ .

$$M = M_Q + M_A \tag{60}$$

$$= p_{q} I_{n} p_{j} H^{*} Q H$$
 (61)

where I<sub>n</sub> is an n<sup>th</sup> order identity matrix

M is given by

$$M^{-1} = \frac{1}{p_{q}} \left[ I_{n} - \frac{p_{J}}{p_{q} + np} \right] + Q H$$
 (62)

This fact can be verified by direct multiplication and recognition that  $H^* = H^{-1}$ .

The optimum weight vector is found from equations (55) and (62).

$$W_{\text{opt}} = \left[I_{n} - \frac{p_{J}}{p_{q} + np_{J}} H * Q H\right] W_{q}$$
 (63)

$$W_{\text{opt}} = W_{q} - \frac{p_{J}}{p_{q} + np_{J}} + W_{q}$$
 (64)

The sequence of matrix multiplications in the last term of equation (64) is carried out as follows:

$$H W_{q} = \begin{bmatrix} \alpha_{1} \\ \alpha_{2} & \exp[-j(\beta_{J} - \beta_{S})] \\ \vdots \\ \alpha_{n} & \exp[-j(n-1)(\beta_{J} - \beta_{S})] \end{bmatrix}$$
(65)

Recalling the expression for  $G_q(\chi)$  from equation (50),

$$Q H W_{q} = G_{q}(\beta_{J}) \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n}$$

$$(66)$$

Finally,

$$H^* Q H W_{q} = G_{q}(\beta_{J}) \begin{bmatrix} 1 \\ \exp[+j\beta_{J}] \\ \exp[+j2\beta_{J}] \\ \vdots \\ \exp[+j(n-1)\beta_{J}] \end{bmatrix}$$

$$= G_{q}(\beta_{J}) B_{J}^{*}, \qquad (68)$$

where  $B_{j}$  is defined similarly to B in equation (53),

$$B_{J} = \begin{bmatrix} 1 \\ \exp[-(j\beta_{J})] \\ \exp[-(j2\beta_{J})] \\ \vdots \\ \exp[-j(n-1)\beta_{J}] \end{bmatrix}$$
(69)

Therefore the optimal weight vector is

$$W_{\text{opt}} = W_{q} - \frac{P_{J}}{P_{q} + nP_{J}} G_{q}(\beta_{J})B_{J}^{*} \qquad (70)$$

Using this weight vector the antenna pattern is given by

$$G(\beta) = B_T W_{opt}$$
  $p_J G_q(\beta_J)B_T B_J^*$  (71)

However, from equation (52)

$$B_{T} W_{Q} = G_{Q}(\beta) . \tag{73}$$

Furthermore,

$$B_{T}B_{J}^{*} = C(\beta - \beta_{J}) \tag{74}$$

where

$$C(x) = \exp j \frac{(n-1)\chi}{2} \frac{\sin (n\chi/2)}{\sin (\chi/2)}$$
, (75)

and 
$$x = (\beta - \beta_j)$$
 (76)

Following this notation,  $G(\beta)$  is given as

$$G(\beta) = G_{q}(\beta) - \frac{p}{P_{q} - nP_{J}} G_{q}(\beta_{J}) C(\beta - \beta_{J})$$
 (77)

This result indicates that when the adaptive loops converge, the resultant antenna voltage pattern is comprised of the sum of two patterns, the quiescent pattern,  $G_q(\beta)$ , and the pattern generated by the weight vector,  $\hat{U}$ . This second pattern is of the form,

$$\frac{\sin K_X}{\sin x}$$
,

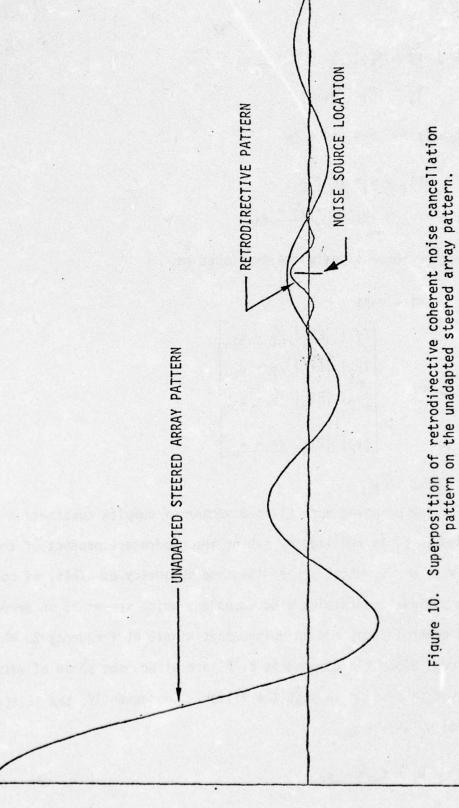
and is centered on the jammer bearing,  $\beta_J$ , as shown by the argument of C. Furthermore, this second pattern is seen to be subtracted from the first and consequently reduces the gain of the array in the direction of the jammer. The pattern of the vector  $\hat{U}$  is seen superimposed on that of S\* in Figure 10. This figure clearly demonstrates the mechanism of coherent noise reduction.

# B. Control Loop Signal Phase Analysis and System Modification - Monochromatic Case

The phase characteristics described in the example of Section II are readily observable from the equivalent schematic of Figure 9. S\* is the steering vector applied to the voltage vector, V, of the array elements. The low pass filters are initialized to zero. The output of the principal summer, El, is given by

$$E1 = V_{\mathsf{T}} S^* \tag{78}$$

and remains constant so long as the input, V, remains constant. Thus, El can be represented as a phasor as can each of the  $v_i$ .



$$E1 = |E1| \underline{/\theta} \tag{79}$$

$$v_{i} = |v_{i}| / / / / / (80)$$

The signals Fl; are given by

$$FI_i = v_i^* |EI| / \theta$$
  
=  $|v_i^*| |EI| / \theta + \phi_i^*$  (81)

As a vector these signals are expressed as

$$F1 = V*E1 \tag{82}$$

F1 = 
$$\begin{bmatrix} |v_1| & |E1_1| & /\theta - \phi_1 \\ |v_2| & |E1_2| & /\theta - \phi_2 \\ |v_3| & |E1_3| & /\theta - \phi_3 \\ \vdots \\ |v_n| & |E1_n| & /\theta - \phi_n \end{bmatrix}$$
(83)

where  $\angle -\phi_i = \angle \phi_i^*$ .

For the purposes here Fl is a vector of complex constants. In the time domain Fl is realized by taking the quadrature product of the two signals El and  $\mathbf{v_i}$  which are of the same frequency  $\omega$ . This, of course, yields a signal containing a dc component which serves as an error signal to the control loop, and an unimportant signal at frequency  $2\omega$  which is filtered. Since the components of Fl are at dc, the phase of each is preserved in passing through the filter. Consequently, the initial phase angle of  $\hat{\mathbf{u_i}}$  is  $\theta$  -  $\phi_i$ ,

$$\hat{\mathbf{u}}_{\mathbf{i}} = \mathbf{C}_{\mathbf{j}} / \theta - \phi_{\mathbf{j}} \tag{84}$$

where  $C_i$  is a real number. Each  $u_i$  serves as a weight on  $v_i$  at the secondary summer. The individual products are

$$\hat{\mathbf{u}}_{\mathbf{i}} \quad \mathbf{v}_{\mathbf{i}} = \mathbf{C}_{\mathbf{i}} | \mathbf{v}_{\mathbf{i}} | \underline{\mathbf{v}}_{\mathbf{i}} + (\mathbf{\theta} - \mathbf{\phi}_{\mathbf{i}})$$

$$= \mathbf{C}_{\mathbf{i}} | \mathbf{v}_{\mathbf{i}} | \underline{\mathbf{v}}_{\mathbf{0}} . \tag{85}$$

Thus each signal entering the secondary summer has phase angle 0 and consequently so does the output E2. The situation at F2; is identical to that at F1; in that each F2; is the product of two sinusoids, one of with phase angle 0 and the other with angle  $\phi_i^*$ . Again the resultant phase angle  $(\theta-\phi_i)$  is preserved through the filter. It is now clear that each  $\hat{u}_i$  remains at the phase angle  $(\theta-\phi_i)$ . Another important fact to be noted from this analysis is that the output of the entire array also maintains a constant phase angle, 0. This further means that each  $z_i$  and  $u_i$  of Figure 3 maintains a constant phase angle  $(\theta-\phi_i)$ . That the  $u_i$  maintain constant phases is of significance since these phase angles are the set of angles initially available from Z in this adaptive scheme. This latter fact is important in that now we can demonstrate that for each  $z_i$  and  $u_i$ ,

$$u_i = k z_i$$
  $i = 1, n$  (86)
final

where k is the same real number for all i.

Considering the phase aspect first, the fact that the corresponding elements of Z and U maintain the same phase angles has already been demonstrated. The initial and final values of the array output may be

represented by  $|E_{in}|/6$  and  $|E_f|/6$ , respectively. Thus, the initial and final values of each  $z_i$  will be

$$(z_i)_{initial} = |E_{in}||v_i| \underline{/\theta - \phi_i}$$
 (87)

and

$$(z_i)_{final} = |E_f||v_i| / (\theta - \phi_i)$$
 (88)

Dividing (88) by (87),

$$(z_i)_{final} = \frac{|E_f|}{|E_{in}|} (z_i)_{initial}$$
 (89)

Each  $z_i$  changes by the same real ratio k where

$$k = \frac{|E_f|}{|E_{in}|}$$
 (90)

Equation (34) describes the response of the integrator in the control loop of Figure 3. In steady state,  $\dot{U} = \dot{W} = 0$  and consequently

$$(u_i)_{final} = (z_i)_{final}.$$
 (91)

Insertion of equation (89) into (91) shows that for each i

$$(u_i)_{final} = k(z_i)_{initial}$$
 (92)

where

$$k = \frac{|E_f|}{|E_{in}|} \tag{93}$$

This is the desired relation.

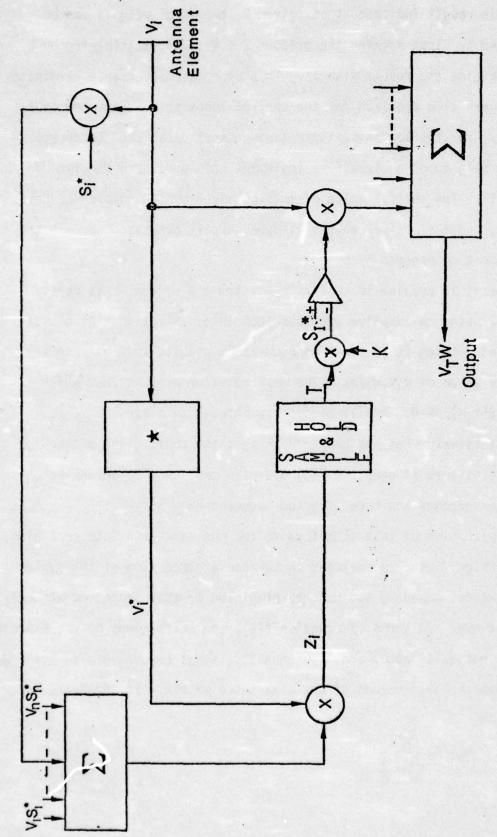
This result indicates that, given k, the final weights can be generated by first forming the product  $Z = V*(V_TW)$ , multiplying by k, and combining the result with S\*. This procedure in essence eliminates the lengthy time required for the control loops to converge, or equivalently, for the low pass filter (integrator) capacitors to charge. A block diagram of a circuit to implement this procedure is given in Figure 11. The initial value of Z is obtained by the sample and hold circuit, and the optimal weight is subsequently generated through the multiplication process by k.

The final problem is that k is not known a priori. k is derivable, however, using an adaptive gradient technique. Starting k at a small value and letting it increase monotonically results initially in a negative value of  $d(V_TW)/dk$ . The desired value of k is that which yields  $d(V_TW)/dk=0$ . k will always lie between zero and one.

Effectively what has happened is that the coefficient of the exponential term of equation (45) has been made zero by adjusting  $\omega_{\mathbf{n_i}}$ . Thus, convergence has been effected independently of any time constant.

Application of this technique to the non-zero bandwidth case also effects significant improvement in the convergence time of the system. In this case, however, the weights predicted by this procedure are only near optimum. In terms of equation (45), the coefficient of the exponential is not quite zero, and, consequently, final convergence is still dependent upon the eigenvalues of the covariance matrix. If, however, the condition

$$\omega_{ni} = \frac{s_n^*}{\lambda_n + \frac{1}{g}}$$



Block diagram of an adaptive control loop incorporating the adaptive initialization modification. Zero bandwidth case. Figure 11.

can be approximated closely, then the system will be very nearly optimized initially, and the effects of poorly behaved eigenvalues will be much less significant.

A control loop designed to take advantage of the phase and amplitude relationships developed above is shown in Figure 12. The functionals inside the dashed line embody the system modification and behave in the following manner. The output of the sample and hold is initialized to zero. The array output forms the quiescent pattern from the steering weights, S\*, and the product  $V*(V_TW)$  is formed as before. Note again that this value is constant in the monochromatic case. The sample and hold circuit obtains a complex sample, T, of the average value of Z. Once the signal vector T has been acquired, the k function, which monotonically increases the scalar multiplier, k, from 0 toward 1, is enabled. Initially, a negative gradient exists on the output of the envelope detector and is measured by a derivative circuit. The output of the derivative circuit permits the k function to continue to operate as long as the negative gradient persists. At some time during the process, however, d(V<sub>T</sub>W)/dk will procede to pass through zero and become positive, at which time the gradient circuit will disable the k function. The final value of the signal vector kT, obtained in this manner, is used to initialize the control loop integrators as shown in the figure. At this point the array elements will have converged to their optimum values. Finally, with the array in it's optimum state, the initialization circuit is switched out of the control loop, and the adaptive array will commence operation in the conventional manner.

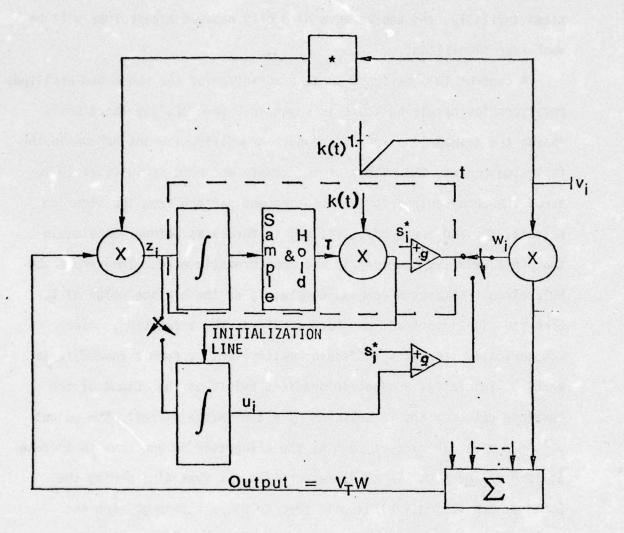


Figure 12. Block diagram of a general adaptive control loop incorporating adaptive initialization for narrow bandwidth coherent noise cancellation.

### IV. ADAPTIVE SYSTEM SIMULATION

A. Adaptive Simulation Incorporating the Gradient Procedure in the Case of Non-Zero Bandwidth

The principles embodied in the theory of the preceeding section may be applied to a system of non-zero bandwidth. Strictly speaking, the weights predicted by a single application of the gradient procedure will not be optimum under the condition of non-zero bandwidth. However, if the jammer bandwidth is sufficiently narrow, then the set of weights generated will provide a significant improvement in the signal-to-noise ratio of the system, and generally, after several iterations of the procedure, initialization to within 3 dB of optimum is effected. Concerning the iterations, the procedure requires that on the second and subsequent iterations, the steering weights, which in the conventional adaptive array remain constant, are updated to the values of the current adapted weights, w, except for a reduction by a factor of the summer gain, g. With the initial advantage so obtained, the system then proceeds toward the optimal configuration in the conventional manner with, consequently, a significant improvement (reduction) in the transient response time. Computer simulation was employed to verify these results.

As an example, an 18 element array was simulated using three discrete noise sources spatially distributed randomly, but restricted to lie outside the mainbeam of the array. The optimal signal-to-noise ratio improvement was found to be 31 dB. Figure 13 indicates, as a function of

noise bandwidth, the improvement in the signal-to-noise ratio that was realized after the initial application of the procedure over the signalto-noise ratio resulting from the unadapted steering weights. Figure 14 shows that for the case of a 40kHz noise bandwidth, although a 16 dB improvement in the signal-to-noise ratio was effected after the initial iteration, the resulting convergence time was not significantly reduced. There are several reasons for this fact. First, an improvement of 16 dB out of a total of 31 dB is only an improvement of about 40 out of 1200, linearly. Secondly, for a given summer gain, g, and filter time constant, τ, the eigenvalues of the noise covariance matrix determine the time constants in the exponential expressions describing the element weight trajectories. These time constants are not altered by this procedure. Consequently, due to the nature of exponential convergence, the closer the system is initialized to the optimum state, the slower is the rate for final convergence. Practically speaking, to achieve a significant factor of improvement in the convergence time, initialization to within 4 or 5 dB of optimum is required. As stated earlier, to achieve this level of improvement, multiple applications of the initialization procedure are required. Continuing the example, additional improvements of 9 and 3 dB were effected on the second and third iterations, respectively. Thus, in this case, initialization to within 3 dB of optimum was effected after three iterations.

Further simulation results were obtained for these three noise sources when the noise bandwidth was changed over the range shown in Figure 13. In almost all cases tested, convergence to within 3 dB was effected within four iterations. Fifth order iterations rarely produced improvement of any consequence.

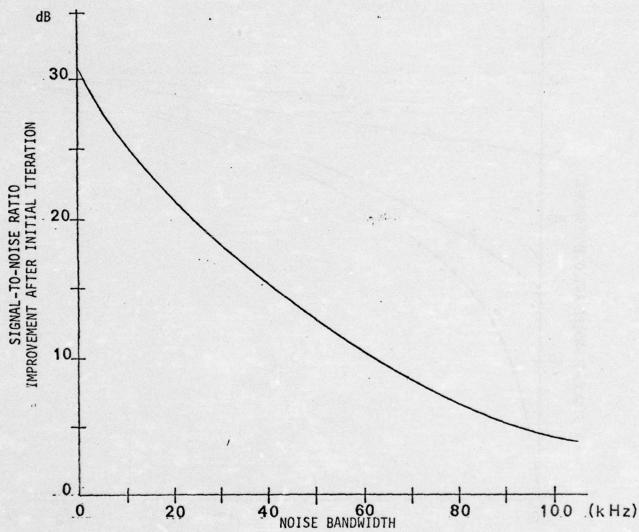


Figure 13. Improvement in signal-to-noise ratio after the first iteration of the adaptive initialization procedure as a function of coherent noise bandwidth. Optimum improvement after convergence- 31 db.

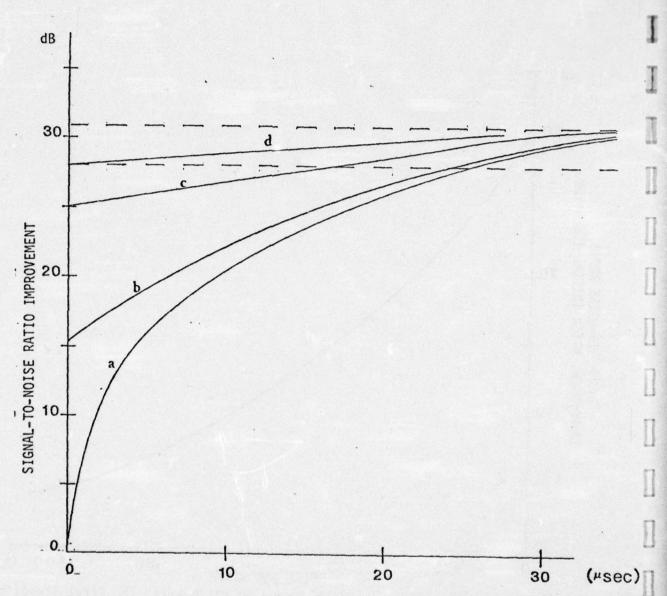


Figure 14. Comparison of convergence time of an adaptive array with:

- a) no adaptive initialization.
  b) one iteration of adaptive initialization.
  c) two iterations of adaptive initialization.
  d) three iterations of adaptive initialization.

In a conventional Howells-Applebaum control system, a poor convergence rate occurs when the jammer-to-noise ratio is relatively small [24]. The gradient procedure, however, generates the initial weight improvement almost independently of the jammer-to-noise ratio. This is due to the action of the multiplier, k. Again, however, note that once the initial improvement is achieved, the same small exponential time constant applies for the remainder of the convergence process. Nevertheless, initialization from 3 to 5 dB of optimum was still observed usually after 3 or 4 iterations.

## B. Broadband Applications

Fundamentally, the problem associated with using the gradient technique lies in the fact that phase stability is required during the period the multiplier, k, is active. This cannot be accomplished if the noise bandwidth is too broad. As long as signal phase shifts can be kept small while the multiplier is active, then local signal-to-noise ratio minima (saddle points) will be avoided. If the bandwidth is increased beyond approximately 75KHz., then the signal-to-noise ratio improvement is not effective. For this reason the gradient procedure would be most effective in CW applications where bandwidths would be consistant with those of Figure 13. For example, if the multiplier function requires 1 µsec. to operate, and a 5% phase shift can be tolerated, then in theory, the limit to noise bandwidth would be approximately 50KHz. This matches well the degradation of the signal-to-noise ratio improvement observed in the simulation (Figure 13).

## V. CONCLUSIONS

A method of generating exactly the optimum weight vector in an adaptive array antenna system has been developed and proven in closed form for the case of zero bandwidth. Extension of this procedure to the case of narrowband coherent noise has been effected using a computer simulation.

By utilizing an interactive adaptive initialization technique, the control loop low pass filters of the conventional Howells-Applebaum implementation of the adaptive array antenna can be initialized such that the signal-to-noise ratio of the array output is generally within 3 or 4 dB of the optimum value. Final convergence is carried out in the conventional fashion. In light of the bandwidth constraints imposed on the system, this method of accelerating array convergence is limited to applications in which the noise bandwidth is less than approximately 75 kHz. In particular, this suggests applications in some CW radar systems.

## **APPENDIX**

The following program simulates the adaptive initialization procedure described in this report. Specifically, the program generates, adaptively, the initial weight vector, applies these weights to the Applebaum control loop and, finally, simulates the trajectory of the Applebaum control loop to the optimum state. In addition to the listing, an example of a data set is included.

This program was run on an IBM 370 machine using G level

Fortran language. It is noted that this particular listing of the

program was made on a CDC 7600. As such, the symbol \( \neq \) seen in this

listing should be recognized as the apostrophe symbol (') of the

IBM code.

THE STEERING WEIGHTS OF THE ADAPTIVE CONTROL LOOP PROPOSED BY APPLEBAUM. THIS PROGRAM SIMULATES AN ANALOG PROCEDURE FOR ADAPTIVELY INITIALIZING //ABANDTWO JOB (FE351,PAGES=44.TIME:-30), #9V ADAPTIVE#,MSGLEVEL=1 STANG = ANGLE AT WHICH THE ANTENNA IS STEERED (IN DEGREES). AFTER INITIALIZATION THE PROGRAM SIMULATES THE CONVERGENCE DISTANCE BETWEEN ANTENNA ELFMENT (IN WAVELENGHTS). PHSRNG = RANGE OF PHASE SHIFT-DETERMINED BY BANDWIDTH THETAI(I) = ELECTRICAL ANGLE OF NOISE SOURCE. TAU =TIME CONTANT OF THE LOW PASS FILTER. TO THE FINAL, OPTIMUM S/N RATIO VALUE NUMBER OF FAR FIELD NOISE SOURCES NUMBER OF ANTENNA ELEMENTS. TIME = MAXIMUM TIME OF OBSERVATION. M(I) = MAGNITUDE OF NOISE SOURCE. PHI(I) = SPATIAL (AZMITH) ANGLE PERINC = PERIOD OF INTEGRATION //STEP1 EXEC FORTGCLG. REGION=120K = GAIN OF AMPLIFIER T = TIME INCREMENT. //SYSPRINT OD DUMMY //FORT.SYSIN DD \*

REAL AIMAGU(K), AIMAGZ(K), AIMAGS(K), AIMAGW(K), WZ(K), WWZ(K) REAL REALU(K) , REALZ(K) , REALS(K) , REALW(K) , WI (K) , WWI (K) COMPLEX ANTE(K), V(K), VCONG(K), U(K), W(K), WV(K), Z(K) 1) CHANGE THE DIMENSIONING STATEMENTS AT THE START OF THE DOUBLE PRECISION THETA3(K), THETA4(K) COMPLEX SIGULT (K) , SV (K) , NV (K) DOUBLE PRECISION EWLI(K,1) COMPLEX SCONG(K), ZINIT(K) DOUBLE PRECISION EWL (K+1) PROGRAM TO SIZE INDICATED. REAL \*8 M3(K) , MM(K) TO RUN PROGRAM:

```
0004000
                                                                                                                                                                                                                                                                                                                                                                                                                                00003200
                                                                                                                                                                                                                                                                                                                                                                                                                                                00003300
                                                                                                                                                                                                                                                                                                                                                                                                                                                                 00003400
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  00003200
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  00003600
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   00003700
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     00003900
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     00004100
                                                                                                                                                                                                                                                                                                                                                                                                                           REAL REALU(18), REALZ(18), REALS(18), REALW(18), W1(16), WW1(18)
REAL AIMAGU(18), AIMAGZ(18), AIMAGS(18), AIMAGW(18), WZ(18), WWZ(18)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  COMPLEX ANTE(18) . V(18) . VCONG(18) . U(18) . W(18) . X(18)
                                                                                                                                                                                                                                                                                                                                                              #I# NOISE SOURCES ARE ON DATA CARDS.
            PRECISION PHI(I), THETAI(I), THETAZ(I)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      DOUBLE PRECISION PHI (3), THETAI (3), THETAZ (3), DLTPH (3)
                                                                                 IN THE FOLLOWING ORDER;
                               PRECISION RAN(I), DLTPH(I)
                                                                                                                                                                                                                                        M(1), THETA1(1), PHI(1)
                                                                                                                                                                                                                                                       M(2), THETA1(2), PHI(2)
                                                                                                                                                                                                                                                                        M(3), THETA1 (3), PHI (3)
                                                                                                                                                                                                                                                                                                                                             -- M(I), THETA1(I), PHI(I)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  DOUBLE PRECISION THETA3(18), THETA4(18)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                COMPLEX SIGVLT(18).SV(18).NV(18)
REAL*8 M(3)
                                                COMPLEX C(I) + ASUM(I)
                                                                                                                                                                                       T, TAU. TIME
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   PRECISION EWL1 (18.1)
                                                                                                                                                                                                        PHSRNG
                                                                                                                                                                                                                         PERINC
                                                                                                                                                                       STANG
                                                                                                                                                                                                                                                                                            *
                                                                                                                                                                                                                                                                                                           ##
 MCI)
                                                                                    DATA CARDS
                                                                                                                                                                                                                                                                                                                                                                                                                                                                  REAL *8 - M3 (18) , MM (18)
                                                                                                                                                                                                                                                                                                                                                             CONTINUE TILL
REAL *8
               DOUBLE
                               DOUBLE
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       DIMENSION RAN(3)
                                                                                                                                                                                                                                                         CARD
                                                                                                                                                                                                                                                                          CARD
                                                                                                                                                                                                                                                                                                           CARD
                                                                                                                                                                                                                                                                                           CARD
                                                                                                                     CARD
                                                                                                                                      CAPD
                                                                                                                                                       CARD
                                                                                                                                                                                        CARD
                                                                                                                                                                                                        CARD
                                                                                                                                                                                                                         CAPO
                                                                                                                                                                                                                                         CARD
                                                                                                                                                                                                                                                                                                                            CARD
                                                                                                    CARD
                                                                                                                                                                        CARD
                                                                                    PUT
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   DOUBLE
                                                                                    S
```

00000

U

COMPLEX PRECISION STANGENT PROPERTY PRO	COMPLEX	C(3) + ASUM (3)	00004500
COMPLEX ANTEV-COMPLX COMPLEX ANTEV-COMPLX CONTINUE  ON IN US  ON US	. 0.	RECISION STANG.PI, ANG. DCOS, AAA, BBB. DSIN	00004400
CONTINUE PI = 3.4159 NPOR = 0 NPOR = 0 AMAX = 0.666	COMPLEX*	ANTEV, CMPLX, SUM 16 DCMPLX	00004200
NFOR = 9 NANY = 0 AMAX = 0. AMAX = 0. GGS = 1. GAA = 60.2 IPUNCH = 0 AINC = 0.2 IFLAGE = 0 INTYPE 0 FIRST = 0. FIRST = 0. FIRS	SS CONTINUE	415	00004700
NPNP = 9.  AMAX = 0.  GGS = 0.  GAA = 60.2  IDUNCH = 0.  INC = 0.2  INC = 0.2  INTLAGE = 0  INTL	110		00007800
AMAX = 0.  GGS = 0.  GA = 60.2  IPUNCH = 0  AINC = 0.2  IFLAGE = 0  IND = 1  GGG = 0.2  IANTVP=0  IFINAL= 0  FEAD 10.1  READ 10.1  READ 10.5  FORMAT (2F20.7)  READ 13.PHSRNG  FORMAT (3E20.7)  READ 13.PHSRNG  FORMAT (2F20.7)	11		00000000
GGG= 0.0 GAA = 60.2 IPUNCH = 0.2 IFLAGE = 0 IFLAGE = 0 IND = 1 GG = 0.2 IANTVP=0 FIRST=0. FIRST=0. IFINAL= 0 FEAD 10.1 READ 10.1 FCRMAT (2F20.7) READ 11. STANG FCRMAT (3F20.7) READ 13.PHSRNG FORMAT (3F20.7) READ 13.PHSRNG FORMAT (3F20.7) READ 13.PHSRNG FORMAT (3F20.7) READ 13.PHSRNG FORMAT (3F20.7) READ 13.PHSRNG FORMAT (2F20.7) READ 13.PHSRNG FORMAT (2F20.7) READ 13.PHSRNG FORMAT (2F20.7)	ni		00000000
ONE = 1 : 00	-		
GAM = 60.2 IPUNCH = 0 INCH = 0 IND = 1 GGG = 0. INTVP=0 FIRST=0. IFINAL= 0 FRAD 10.1 READ 10.1 READ 11.5 FORMAT(3110) READ 11.5 FORMAT(320.7) READ 11.5 FORMAT(320.7) READ 11.5 FORMAT(320.7) READ 13.9 FORMAT(320.7) READ 13.9 FORMAT(320.7)	- '		00004800
1PUNCH = 0 4INC = 0.2 IFLAGE = 0 IDD = 1 6GG = 0. IANTVP=0 FIRST=0. IFINAL = 0 READ 10.1 READ 10.7 READ 10.7 READ 11. STANG FORMAT (3F20.7) READ 12. T. TAU.TIME FORMAT (3F20.7) READ 13.PERINC READ 13.PERINC	09		0004400
IFLAGE = 0	· <		0000000
IFLAGE = 0 100 = 1 660 = 0 1ANST = 0 FIRST = 0 FINAL = 0 FEAD 10.1 READ 10.4 FORMAT(2F20.7) READ 11.6 FORMAT(3F20.7) READ 13.PERINC FORMAT(3F20.7) READ 13.PERINC FORMAT(3F20.7) READ 13.PERINC FORMAT(3F20.7) READ 13.PERINC READ 13.PERINC	) I	•	0000000
GGG = 0. GGG = 0. IANTVP=0 FIRST=0. FINAL= 0 READ 10.4 FORMAT(3110) READ 10.5 FORMAT(3F20.7) READ 11. STANG FORMAT(3F20.7) READ 13.PERINC READ 13.PERINC			0015000
GGG = 0. IANTVP=0 FIRST=0. FIRST=0. FEAD 10.1 READ 10.1 READ 20.0. FORMAT(2110) READ 11. G FORMAT(3F20.7) READ 12. T. TAU.TIME FORMAT(3F20.7) READ 13.PERINC FAD 13.PERINC READ 13.PERINC			00005500
IANTVP=0 IANTVP=0 FIRST=0.  IFINAL= 0 READ 10.1 READ 10.1 READ 200.D FORMAT(2F20.7) READ 11. STANG FORMAT(3F20.7) READ 13.PERINC READ 13.PERINC READ 13.PHSRNG FORMAT(E20.7) READ 11. FIRETAl(N).PHI(N)	0		00002300
FIRST=0.  IFINAL= 0  READ 10.1  READ 10.5  READ 200.0  FORMAT(2F20.7)  READ 11. G  FORMAT(3F20.7)  READ 12. T.TAU.TIME  FORMAT(3F20.7)  READ 13.PERINC  READ 13.PERINC  READ 13.PHSRNG  FORMAT(E20.7)  READ 13.PHSRNG  FORMAT(E20.7)  DO 1 N = 1.1  READ 11.M(N).THETAI(N).PHI(N)	111		
IFINAL = 0 READ 10.1 READ 10.1 READ 20.0 FORMAT(2110) READ 20.0 FORMAT(2F20.7) READ 11.6 FORMAT(3F20.7) READ 12. T. TAU, TIME FORMAT(3F20.7) READ 13.PERINC	0		
READ 10.1  READ 10.K  FORMAT (3110)  READ 200.D  FORMAT (2F20.7)  READ 11. G  PEAD 11. G  PEAD 11. STANG  FORMAT (3F20.7)  READ 12. T.TAU.TIME  FORMAT (3F20.7)  READ 13.PERINC  READ 13.PERINC  READ 13.PHSRNG  FORMAT (E20.7)  DO 1 N = 1.1  READ 11.M(N).THETA1(N).PHI(N)	11		
READ 16.K FORMAT (3116) READ 260.D FORMAT (2F20.7) READ 11. STANG FORMAT (3F20.7) READ 12. T.TAU.TIME FORMAT (3E20.7) READ 13.PERINC READ 13.PHSRNG FORMAT (E20.7) DO 1 N = 1.1	ċ		0002000
FORMAT (3110)  READ 200.D  FORMAT (2F20.7)  READ 11. G  PEAD 11. STANG  FORMAT (3F20.7)  READ 12. T.TAU.TIME  FORMAT (3E20.7)  READ 13.PERINC  READ 13.PHSRNG  FORMAT (E20.7)  OO 1 N = 1.1  READ 11.M(N).THETA1(N).PHI(N)	6	×	00002100
READ 200.0 FORMAT (2F20.7) READ 11. G PEAD 11. STANG FORMAT (3F20.7) READ 12. T.TAU.TIME FORMAT (3F20.7) READ 13.PERINC READ 13.PHSRNG FORMAT (E20.7) DO 1 N = 1.1	FORMAT (3	1	00002800
FORMAT (2F20.7)  READ 11. G  PEAD 11. STANG  FORMAT (3F20.7)  READ 12. T.TAU.TIME  FORMAT (3F20.7)  READ 13.PERINC  READ 13.PHSRNG  FORMAT (E20.7)  DO 1 N = 1.1  READ 11.M(N).THETA1(N).PHI(N)	500	÷	0002300
READ 11. G  PEAD 11. STANG  1 FORMAT (3F20.7)  READ 12. T. TAU. TIME  2 FORMAT (3F20.7)  READ 13.PERINC  READ 13.PHSRNG  3 FORMAT (E20.7)  DO 1 N = 1.1  READ 11.M(N).THETA1(N).PHI(N)	FORMAT (2	F2	00090000
PEAD 11. STANG 1 FORMAT (3F20.7) READ 12. T. TAU.TIME 2 FORMAT (3F20.7) READ 13.PERINC READ 13.PHSRNG 3 FORMAT (E20.7) DO 1 N = 1.1 READ 11.M(N).THETA1(N).PHI(N)	-		0000000
1 FORMAT (3F20.7)  READ 12. TitAU.TIME 2 FORMAT (3E20.7)  READ 13.PERINC  READ 13.PHSRNG  READ 13.PHSRNG  3 FORMAT (E20.7)  DO 1 N = 1.1  READ 11.M(N).THETA1(N).PHI(N)	-	STANG	00009000
READ 12. T.TAU.TIME 2 FORMAT(3E20.7) READ 13.PERINC READ 13.PHSRNG 3 FORMAT(E20.7) DO 1 N = 1.1 READ 11.M(N).THETA1(N).PHI(N)	[3	F20.7)	00099000
2 FORMAT (3E20.7)  READ 13.PERINC  READ 13.PHSRNG  3 FORMAT (E20.7)  DO 1 N = 1.1  READ 11.M(N).THETA1(N).PHI(N)	12.	T, TAU, TIME	000990000
READ 13.PERINC READ 13.PHSRNG 3 FORMAT(E20.7) DO 1 N = 1.1 READ 11.M(N).THETA1(N).PHI(N)	FORMAT (3	E20.7)	00099000
READ 13.PHSRNG 3 FORMAT(E20.7) DO 1 N = 1.1 READ 11.M(N),THETA1(N),PHI(N)	13,	PERINC	
3 FORMAT(E20.7)  DO 1 N = 1 · I  READ 11 · M(N) · THETA1(N) · PHI(N)	3	PHSRNG	
$V_1 = 1 \cdot 1$ 11 · M(N) · THETA1 (N) • PHI (N)	3 FORMAT (E	20.7)	
1.• K(Z) • THETA1(Z) • DHI(Z)	-	1.1	000990000
	-	-	00090000

00008800	00007100 00007100 0000720	*31X* # (IN VOLTS) # 19X* # (IN DEGREES) # 18X* # (IN DEGREES) # ) 00007500 *50(28X*F10.3*21X*F10.3*20X*F10.3/# #)) 00007600 ANG*PI/180.	00008400 00008500 00008600	00008800 00008900 00009000	00009300	00009600 00009700 00009800	0001000
ō	3666	(S) # (S)		6666	666	8666	õ
		DEGREE					
		NI)#					
	*:	# 18X					!
	1,1) TAGE#,10	GREES)		*.#·			
W X	N = 1,	(IN DE				i digital	
I•K•D•G•STANG•T•TAU•TIME RINC•PHSRNG (360•*PERINC))*2• T*360•	(M(N), THETAI(N), PHI(N), N, 1/4 #, 26x, #NOISE SOURCE V	,19X,#	(e)			ITMCT= #.14)  ITMO ING I ING ITMO ITMETAL (N) *3.14159/180	
I • K • D • G • STANG • T • ER INC • PHSRNG • * 7 (360 • * PER INC) ) * 2 / 1 * 360 •	1 (N) P	3,21X			. 9	#.*ITMCT= #.14) TMCT STANG 1,1 PHI(N)*3.14159/180. = THETA1(N)*3.1415	N(I+K+D+EML+PHI)
PHSRNG*PERIN	THETA	X, F10	20*0*	68	DCOS(ANG)/G DSIN(ANG)/G IC/T+ 0.1	T= #.14) 3.14159/	EALU.
1.1	(M(N),	*31X*#(IN V *50(28X*F10 ANG*PI/180.	*********	N N X	= DSIN (AN NC/T+ 0.1	****ITWCT STANG 1*1 PHI(N)*	N(I,K,D,EWL,PHI)
105. 105. 14.8 HSRNC	108 107 107	## 1		. AAO	REALS(N) = AIMAGS(N) = CONTINUE ITMCT=PERI		MA
CONTI PRINT PRINT BW= (P PHSHF	•		WWZ (N)	AA	REALS(N) AIMAGS(N) CONTINUE ITMCT=PEF		פרר
	106	108				25	

```
000101000
                                                                                                                                                                                                                                                                                                 00010000
                                                                                                                                                                                                                                                                                                                    00010300
                                                                                                                                                                                                                                                                                                                                                                           0001000
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           000111000
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             00011200
                                                                                                                                                                                                                                                                                                                                                                                              0001000
                                                                                                                                                                                                                                                                                                                                                                                                                 0001000
                                                                                                                                                                                                                                                                                                                                                                                                                                   00010000
                                                                                                                                                                                                                                                                                                                                                                                                                                                     0001000
                                                                                                                                                                                                                                                                                                                                                                                                                                                                         00011000
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               00011300
                                                                                                                                                                                                                                                                            CALL SUMM (I.K.EWL, THETA1, THETA2, THETA3, THETA4, C, ANTE, M, M3)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        #, E20.7,2X, # DB#)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      CALL SIGNOS (1, K, D, PHII, M, V, SNR, EWL 1, SIGVLT, SV, NV)
                                                                                                                                                                                                                                                                                                                                                                                             FORMAT (#1#//+55X+#ANTENNA ELEMENT VOLTAGES#//)
                                                                                                                                                                                                                                                                                                                   CALL RECTAZ(I,K,M3,MM,THETA3,THETA4,V,VCONG)
IF (IANTVP .EO. 1) GO TO 47
                                                                                                                                                                                                                                                                                                 CONG(I+K+D+M3+MM+THETA3+THETA4)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         FORMAT (#0# + # SIGNAL TO NOISE RATIO =
              CALL PHAGEN(I, PHSRNG, PH, AITMCT, RAN)
                                                                                                                                                                RAN (N) = (RAN (N) /A) * (PHSRNG*PI/180.)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                LPF (K.T. TAU, REALU, REALZ, NPNP)
                                                                                                         IF (RAN(N+1) .GT. A) A= RAN(N+1)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            BRAKE (K.Z. REALZ, AIMAGZ)
                                                                                                                                                                                                                                          THETAI (N) = THETAI (N) + RAN (N)
                                                                                                                                                                                                                                                                                                                                                                                                                                                   FORMAT (# #,30X,E30.7,E25.7)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                IF (FIRST .NE. 0.) GO TO 30
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           CALL MULTI (K.SIJM. VCONG.Z)
                                    A= RAN(1)
IF(A .EO. 0.) GO TO 112
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            CALL WEIGHT (K.W.V.WV)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              ADD (K, WV, SUM)
                                                                                                                                                                                                        DO 26NN=1.ITMCT
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      PRINT 1500, SNR
                                                                                                                                                                                                                                                                                                                                                                                                                                  PRINT 40.V(J)
                                                                                         U-111 N=1.0
                                                                                                                                                10 112 N=1.1
                                                                                                                                                                                                                                                                                                                                                                                                               00 39 J= 1,K
                                                                                                                                                                                                                        DO 20 N=1,1
PH=PHI (1)
                                                                                                                                                                                                                                                                                                                                                           I ANTVP= 1
                                                                                                                              CONTINUE
                                                                                                                                                                                     CONTINUE
                                                                                                                                                                                                                                                                                                                                                                                                                                                                       CONTINUE
                                                                                                                                                                                                                                                               CONTINUE
                                                                                                                                                                                                                                                                                                                                                                           PRINT 70
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           CONTINUE
                                                                         [-I =0
                                                                                                                                                                                                                                                                                                    CALL
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   CALL
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                CALL
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 CALL
                                                                                                                                                                                  112
                                                                                                                                                                                                                                                               20
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          1500
                                                                                                                                                                                                                                                                                                                                                                                                                                                                       39
                                                                                                                             111
```

CALL LPF(K.T.TAU). # im #dC2.NPNP)  PRINT 33.				
FORMATIE 404:4U= 4:2E20.7)  CONTINUE CALL BRAKE(1.ASUM.AIMSUM) SMAGKP = SORT(RSUW.AIMSUM) SMAGKP = SORT(RSUW.ex-4.AIMSUM*2) IF(IDD,EO.1) GO TO 213 IDD=2 GAA = GAA-AINC GAA = GAA-AINC AIMGGU(N) = REAL(ZINIT(N))/GAA AIMGGU(N) = REAL(ZINIT(N))/GAA AIMGGU(N) = REAL(ZINIT(N))/GAA AIMGGU(N) = REAL(ZINIT(N))/GAA AIMGGU(N) = CMPLX(REALU'N)/GAA CALL ADD?(X-AIMGGN-AIMGW+W) CALL ADD?(X-AIMGS-AIMGW+W) CALL ADD?(X-AIMGS-AIMGW+W) CALL ADD?(X-AIMGS-AIMGW+W) CALL ADD?(X-AIMGS-AIMGW+W) FORMAT(FO#-3X-F2O.7+10X-2E2O.7) IF(IFINAL EG-1) GO TO 3030 ASUM(1) = SUM SMAGKP = SMAGKP SMAGKP = SMAGKP		CALL LPF (K.T. TAU, AIMAGU, AIMAGZ, NPNP) PRINT 33. REALU(1), AIMAGU(1)		
CALL RRAKE(1.ASUM.AIMSUM) SMAGKP = SORT (RSUW.*AIMSUM) SMAGKP = SRRT (RSUW.*AZHAIMSUM) IF (IDD.EO.1) GG TO 2113 IDD=2  GAA-AINC DG 2118 N = 1.K REALU(N) = REAL(ZINIT(N))/GAA AIMGU(N) = AIMAG(ZINIT(N))/GAA AIMGU(N) = AIMAG(ZINIT(N))/GAA AIMGU(N) = CMPLX(REALU(N)*AIMAGU(N)) CONTINUE GO TO 113 CONTINUE CALL ADD? (K.AIMAGS.AIMAGW.W) FORMAT(#0#.3X*EZO.7*10X*ZEZO.7*10X*ZEZO.7) IF (ITINAL EG. 1) GO TO 3030 ASUM(1) = SUM SMAGKP = SMAGKP SMAGKP = SMAGKP	33	FORMAT(#0#,#U= #.2E20.7)	• 24	
CALL TOGETH(K.REALU.AIMAGU.U)  If (IDD.ECO.1) GO TO 2113  If (IDD.ECO.1) GO TO 2113  If (IDD.ECO.1) GO TO 2113  GAA = GAA - AINC  DO 2118 N = 1.K  REALU(N) = REAL(ZINIT(N)), GAA  AIMAGU(N) = AIMAG(ZINIT(N)), GAA  AIMAGU(N) = CMPLX(REALU(N), AIMAGU(N))  CONTINUE  GO TO 113  CONTINUE  GO TO 113  CALL ADD? (K.AIMAGS, AIMAGU, AIMAGW, G)  CALL ADD? (K.AIMAGS, AIMAGU, AIMAGW, G)  CALL ADD? (K.W.Y.W.Y)  IF (IFLAGF, GE.1) PRINT 2119, GAA.W(1), SUM  IF (IFLAGF, GE.1) PRINT 2119, GAA.W(1), SUM  IF (IFLAGF, GE.1) PRINT 2119, GAA.W(1), SUM  CALL ADD (K.W.Y.SUM)  IF (IFLAGF, GE.1) GO TO 3030  ASUM(1) = SUM  CALL BRAKE(1, ASUM.RSUM.AIMSUM.AIMAGW, G) TO 2900  SMAGST = SMAGKP  SMAGST = SMAGKP	1	ASUM (1) = SUM	00011500	
IF (IDD.EC.1) GG TO 2113  IDD=2  GAA = GAA-AINC  GAA = GAA-AINC  GAA = GAA-AINC  GAA = GAA-AINC  GAB =		SMAGKP = SORT (RSUM**2+AIMSUM**2)	00011700	6.0
IDD=2 GAA-AINC GAA = GAA-AINC DO 2118 N = 1.K REALU(N) = REAL(ZINIT(N)) / GAA AIMAGU(N) = AIMAG(ZINIT(N)) / GAA U(N) = CWPLX(REALU(N), AIMAGU(N)) CONTINUE DO 2114 N=1.K ZINIT(N)=U(N) / G ZINIT(N)=U(N) / G ZINIT(N)=U(N) / G CONTINUE CALL ADDZ(K, REALS, REALU, REALW, G) CALL ADDZ(K, REALS, REALU, REALW, G) CALL ADDZ(K, REALS, REALU, REALW, G) CALL ADDZ(K, AIMAGS, AIMAGW, W) CALL ADDZ(K, W) / W) CALL ADDZ(K, W) / W) CALL ADDX(K, W) / W) CALL ADDZ(K, W) / W) CALL ADDX(K, W) / W) CALL ADDX(K, W) / W) CALL ADDX(K, W) / W) FIF (IFLAGF, GE, 1) PRINT 2119, GAA, W(1), SUM FIF (IFLAGF, GE, 2) PRINT 2119, GAA, W(1), SUM FIF (IFLAGF, GE, 1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1, ASUM, AIMSUM, AIMSUM, SUM(1)) = SUM CALL BRAKE(1, ASUM, AIMSUM, AIM		IF (IDD.EO.1) GO TO 2113	00011900	
DO 211R N = 1.K  REALU(N) = REAL(ZINIT(N)), GAA  AINAGU(N) = AIMAG(ZINIT(N)), GAA  U(N) = CMPLX(REALU(N), AIMAGU(N))  CONTINUE  GO TO 113  CONTINUE  CONTINUE  GO TO 2117  CONTINUE  CONTINUE  CONTINUE  CALL ADD?(K.REALW.AIMAGW.W)  CALL ADD?(K.REALW.AIMAGW.W)  CALL ADD?(K.W.Y.W.V)  CALL ADD?(K.W.Y.W.V)  CALL ADD(K.W.Y.SE20.7)  IF (IFLAGF.GE.1) PRINT 2119. GAA.W(1), SUM  FORMAT(#0#.3X.E20.7)  IF (IFINAL .EO. 1) GO TO 3030  ASUM(1) = SUM  SMG = SORT(RSUM**2 + AIMSUM**2)  IF (SMAG.F)  SMAGST = SMAGKP  SMAGST = SMAGKP  SMAGST = SMAGKP  SMAGST = SMAGKP	2117	IDD=2 GAA = GAA-AINC	00012100	
REALU(N) = REAL(ZINIT(N))/GAA AIMAGU(N) = AIMAG(ZINIT(N))/GAA AIMAGU(N) = AIMAG(ZINIT(N))/GAA U(N) = CMPLX(REALU(N), AIMAGU(N)) CONTINUE GO TO 113 CONTINUE DO 2114 N=1,K ZINIT(N)=U(N)/G CONTINUE GO TO 217 CONTINUE CALL ADD2(K*REALS*REALU*REALW*G) CALL ADD2(K*AIMAGS*AIMAGU*M) CALL ADD2(K*AIMAGS*AIMAGU*M) CALL ADD2(K*AIMAGS*AIMAGU*M) CALL ADD1K*W*SUM F(IFINUE CALL ADD1K*W*SUM F(IFINUE CALL ADD1K*W*SUM F(IFINUE CALL ADD1K*W*SUM F(IFINUAL *EG*1) PRINT Z119* GA*W(1))*SUM F(IFINUAL *EG*1) PRINT Z119* GA*W(1))*SUM F(IFINUAL *EG*1) GO TO 3030 ASUM(1) = SUM ASUM(1) = SUM ASUM(1) = SUM F(IFINUAL *EG*1) GO TO 2900 SMAGST= SMAGKP SMAGKP SMAGKP SMAGKP	2020	DO 2118 N = 1.K	00012200	
AIMAGUIN) = AIMAGUIN) (CAA)  O(NTINUE  GO TO 113  CONTINUE  GO TO 2114  CONTINUE  GO TO 2117  CONTINUE  GO TO 2117  CALL ADD2(K*REALS, REALU*REALW*G)  CALL ADD7(K*W*V*SUM)  IF(IFLAGF, GE-1) PRINT 2119* GAA*W(1)*SUM  IF(IFLAGF, GE-1) PRINT 2119* GAA*W(1)*SUM  FORMAT(#O#*3*, E20.7*10*, 2E20.7*)  IF(IFINAL *E0.1) GO TO 3030  ASUM(1) = SUM  SMAG **SAMAGKP**  SMAGST = SMAGKP  SMAGST = SMAGKP  SMAGKP = SMAGKP		REALU(N) = REAL(ZINIT(N))/GAA	00012300	
CONTINUE 60 TO 113 CONTINUE 60 TO 113 CONTINUE DO 2114 N=1.K ZINIT(N)=U(N)/G ZINIT(N)=U(N)/G ZINIT(N)=U(N)/G CONTINUE CONTINUE CALL ADD2(K.AIMAGS,AIMAGU,AIMAGW,G) CALL ADD2(K.AIMAGS,AIMAGW,W) CALL ADD2(K.AIMAGS,AIMAGW,W) CALL ADD2(K.WV,SUM) IF(IFLAGF.GF.1) PRINT 2119. GAA.W(1).SUM FORMAT(#0#.20.7) OX.2E20.7) IF(IFLAGF.GF.1) PRINT 2119. GAA.W(1).SUM CALL BRAKE(1.ASUM.RSUM.AIMSUM) SMAG = SORT(RSUM**2 + AIMSUM**2) IF (SMAG.GT.SMAGKP) GO TO 2900 SMAGST= SMAGKP SMAGST= SMAGKP		AIMAGU(N) = AIMAG(ZINI) (N)) / CAA	00012500	
GO TO 113  CONTINUE  DO 2114 N=1.K  ZINIT(N)=U(N)/G  CONTINUE  GO TO 2117  CONTINUE  CALL ADD2(K.AIMAGS,AIMAGW,W)  CALL ADD2(K.AIMAGS,AIMAGW,W)  CALL ADD7(K.WV,VWV)  CALL ADD(K.WV,SUM)  CALL ADD(K.WV,SUM)  IF (IFLAGF.GE.1) PRINT 2119. GAA.W(1).SUM  IF (IFLAGF.GE.1) PRINT 2119. GAA.W(1).SUM  IF (IFLAGF.GE.1) PRINT 2119. GAA.W(1).SUM  CALL ADD(K.WV,SUM)  IF (IFLAGF.GE.1) PRINT 2119. GAA.W(1).SUM  CALL ADD(K.WV,SUM)  IF (IFLAGF.GE.1) GO TO 3030  ASUM(1) = SUM  CALL BRAKE(1.ASUM.RSUM.AIMSUM.)  SMAG = SORT(RSUM**2 + AIMSUM**2)  IF (SMAG.GT.SMAGKP) GO TO 2900  SMAGST = SMAGKP  SMAGKP = SMAGKP	2118	CONTINUE	00012600	
CONTINUE  DO 2114 N=1.K  ZINIT(N)=U(N)/G  CONTINUE  CONTINUE  CALL ADD2(K*REALU*REALU*G)  CALL ADD2(K*REALU*AIMAGU*G)  CALL ADD2(K*REALU*AIMAGU*G)  CALL ADD(K*WV*SUM)  CALL ADD(K*WV*SUM)  F(IFLAGF.GE.1) PRINT 2119* GAA*W(1)*SUM  FORMAT(#O≠*3X*E20.7*10X*2E20.7*10X*2E20.7*)  IF(IFINAL .EQ. 1) GO TO 3030  ASUM(1) = SUM  CALL BRAKE(1*ASUM*RSUM*RSUM)  SMAG = SORT(RSUM*R2 * AIMSUM*R2)  IF (SMAG.GT.SMAGKP) GO TO 2900  SMAGST= SMAGKP  SMAGKP	•	60 TO 113	012	
DO 2114 N=1.K ZINIT(N)=U(N)/G CONTINUE GO TO 2117 CALL ADD2(K.REALS, REALU, REALW,G) CALL ADD2(K.AIMAGS,AIMAGW,W) CALL ADD2(K.AIMAGS,AIMAGW,W) CALL ADD(K.WV,WV) CALL ADD(K.WV,SUM) IF(IFLAGF,GE,1) PRINT 2119. GAA,W(1),SUM FORMAT(#O#.3X.E20.7,10X,2E20.7) IF(IFINAL .EA.1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1,ASUM,RSUM,AIMSUM,SMG = SORT(RSUM,*2 + AIMSUM,*2) IF (SWAG,GT,SMAGKP) GO TO 2900 SMAGST= SMAGKP SMAGKP = SMAG	2113	CONTINUE	0012	
CONTINUE GO TO 2117 CONTINUE GO TO 2117 COLL ADD2(K.REALS, REALU.REALW.G) CALL ADD2(K.AIMAGS, AIMAGW.M) CALL TOGETH(K.MEALW.AIMAGW.W) CALL WFIGHT(K.W.V.WV) CALL ADD(K.WV.SUM) IF (IFLAGE.GE.1) PRINT 2119. GAA.W(1).SUM IF (IFLAGE.GE.1) PRINT 2119. GAA.W(1).SUM IF (IFLAGE.GE.1) PRINT 2119. GAA.W(1).SUM SANGE.GE.SUM CALL BRAKE(1.ASUM.RSUM.AIMSUM) SMAG = SORT(RSUM*2 + AIMSUM*2) IF (SMAG.GT.SMAGKP) GO TO 2900 SMAGST = SMAGKP SMAGST = SMAGKP		-	01770	
GO TO 2117  CONTINUE  CALL ADD2(K*REALS*REALU*REALW*G)  CALL ADD2(K*AIMAGU*AIMAGW*G)  CALL ADD2(K*AIMAGU*AIMAGW*G)  CALL WFIGHT(K*W*V*WV)  IF (IFLAGF.GF.1) PRINT 2119* GAA*W(1)*SUM  FORMAT(‡O≠*3X*E20.7*10X*2E20.7*10X*2E20.7)  IF (IFINAL *EQ. 1) GO TO 3030  ASUM(1) = SUM  CALL BRAKE(1*ASUM*R2 * AIMSUM**2)  IF (SMAG*GT.SMAGKP) GO TO 2900  SMAGST= SMAGKP  SMAGKP = SMAGKP	2114		00013100	
CONTINUE CALL ADD2(K*REALS*REALU*REALW*G) CALL ADD2(K*AIMAGS*AIMAGW*G) CALL TOGETH(K*REALW*AIMAGW*W) CALL WFIGHT(K*W*V*WV) CALL WFIGHT(K*W*V*WV) CALL ADD(K*WV*SUM) IF (IFLAGE*GE*I) PRINT 2119* GAA*W(1)*SUM FORMAT(#O#*3X*E20*7*10X*2E20*7) IF (IFINAL *E0* 1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1*ASUM*RSUM*AIMSUM) SMAG = SORT(RSUM**2 * AIMSUM**2) IF (SMAG*GT*SUM**2 * AIMSUM**2) IF (SMAG*GT*SMAGKP) GO TO 2900 SMAGST= SMAGKP		60 TO 2117	00013600	
CALL ADD2(K, AIMAGS, AIMAGW, G) CALL ADD2(K, AIMAGS, AIMAGW, W) CALL TGGTH(K, REALW, AIMAGW, W) CALL WFIGHT(K, W, V, WV) CALL ADD(K, WV, SUM) IF (IFLAGF, GE, 1) PRINT 2119, GAA, W(1), SUM FORMAT(#O#.3X, E20.7, 10X, 2E20.7) IF (IFINAL .E0. 1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1, ASUM, RSUM, AIMSUM) SMAG = SORT(RSUM**2 + AIMSUM**2) IF (SMAG, GT, SMAGKP) GO TO 2900 SMAGST = SMAGKP SMAGKP	113		00013700	
CALL TOGETH(K*REALW*AIMAGW*W) CALL WFIGHT(K*W*V*WV) CALL ADD(K*WV*SUM) IF(IFLAGF.GE.1) PRINT 2119. GAA.W(1).SUM FORMAT(#O#.3X*E20.7*10X*2E20.7) IF(IFINAL .EQ. 1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1.ASUM*RSUM*AIMSUM) SMAG = SORT(RSUM*R2 + AIMSUM**2) IF (SMAG*GT.SMAGKP) GO TO 2900 SMAGST= SMAGKP SMAGKP		CALL ADD2 (K, AIMAGS, AIMAGU, AIMAGW, G)		
T 2119. GAA.W(1).SUM 10X,2E20.7,10X,2E20.7) TO 3030 UM.AIMSUM! + AIMSUM**2) GO TO 2900		CALL TOGETH(K.REALW, AIMAGW, W)	00014000	
IF (IFLAGE.GE.1) PRINT 2119. GAA.W(1).SUM_FORMAT(#0#.3X.E20.7)10X.2E20.7) IF (IFINAL .EQ. 1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1).ASUM.RSUM.AIMSUM) SMAG = SORT(RSUM**2 + AIMSUM**2) IF (SMAG.GT.SMAGKP) GO TO 2900 SMAGST= SMAGKP SMAGKP		CALL ADD (K+WV+SUM)	00014200	
IF(IFINAL .EQ. 1) GO TO 3030 ASUM(1) = SUM CALL BRAKE(1, ASUM, RSUM, AIMSUM) SMAG = SORT(RSUM**2 + AIMSUM**2) IF (SMAG, GT, SMAGKP) GO TO 2900 SMAGST= SMAGKP SMAGKP	27.10	IF (IFLAGE, GE, 1)	00014300	
UM•AIMSUM**2) • AIMSUM**2) 60 TO 2900	2117	IF (IFINAL .EQ.		
4 AIMSUM**2) 60 TO 2900		ASUM(1) = SUM	00014500	
GO TO 2900			00014100	
		20 10	00014800	
		SMAGKP = SMAG	00014900	

```
00015400
00012000
               00015100
                               00015200
                                             00015300
                                                                                                       00015600
                                                                                                                     00015700
                                                                                                                                                                                                                                                                                                                                                                                                                                          00020300
                                                                                                                                                                                                              ±00018800
                                                                                                                                                                                                                              00018900
                                                                                                                                                                                                                                            ±00019000
                                                                                                                                                                                                                                                            000191000
                                                                                                                                                                                                                                                                                                                                                                  ±00019800
                                                                                                                                                                                                                                                                                                                                                                                              ≠00020000
                                                                                                                                                                                                                                                                                                                                                                                                            00020100
                                                                                                                                                                                                                                                                                                                                                                                                                          ±00020200
                                                                                                                                                                                                                                                                                                         ±00019400
                                                                                                                                                                                                                                                                                                                                                                                00019900
                                                                                                                                                                                                                                                                           ==00019200
                                                                                                                                                                                                                                                                                         00019300
                                                                                                                                                                                                                                                                                                                       00019500
                                                                                                                                                                                                                                                                                                                                     =+00019600
                                                                                                                                                                                                                                                                                                                                                   00019700
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       MAXINUM PHASE SHIFT (DETERMINED BY BANDWIDTH) = +/-
                                                                                                                                                                                                                                                                          DISTANCE BETWEEN ANTENNA ELEMENTS (IN WAVELENGHTS)
                                                                                                                                                                                                                                                                                                                                    ANGLE AT WHICH THE ANTENNA IS STEERED (IN DEGREES)
                                                                                                                                                                                                              NUMBER OF FAR FIELD NOISE SOURCES =
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     #,32X,E13,3)
                                                                                                                                                                                CALL SIGNOS (1,K,D,PHI1,W,V,SNR,EWL1,SIGVLT,SV,NV)
                                                                                                                                                                                                                                                                                                                                                                                                                                                         PERIOD OF INTEGRATION =
                                                                                                                                                                                                                                                                                                                                                                                              TIME CONSTANT OF LOW PASS FILTER
                                                                                                                                                                                                                                                                                                                                                                AMOUNT TIME IS INCREMENTED BY
                                                                                                                                                                                                                                                                                                                                                                                                                           MAXIMUM TIME OF OBSERVATION =
                                                                                                                                                                                                                                            NUMBER OF ANTENNA ELEMENTS =
                                                                                                                                                                                                                                                                                                       GAIN OF SUMMING AMPLIFIER =
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     BANDWIDTH=
60 TO 2117
STOP
                                            IF (IFLAGE.EQ.1) GO TO 3000
                                                                                                                                                                                                             FORMAT (#1#///# #9#
                                                                                                                                                                                                                                                                                                                                                                                                                                                         #·//**
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    FORMAT (# #.//#0#.#
                                                                                         SAA= GAA+ 2. *AINC
               GAA.LE. .11)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   JT=(TIME/T)+ 0.1
                                                                                                     = AINC/20.
                                                                                                                                                                                              PRINT 1500, SNR
                                                                          SMAGKP= SMAGST
                                                                                                                                     GAA= GAA+ AINC
GAA .GT.
                                                                                                                                                                                                                                                                                                                                                                                                                                                                        * .10X.E13.3
                                                                                                                      50 TO 2117
                              50 TO 2117
                                                                                                                                                                 60 TO 2020
                                                             FLAGE = 1
                                                                                                                                                  IF INAL=
                                                                                                                                                                                                                                                                                                                                                                                                                                                         FORMAT (*
                                                                                                                                                                                                                             **10X*16*
                                                                                                                                                                                                                                            1/+0++
                                                                                                                                                                                                                                                                          ナ・ナリナノノ
                                                                                                                                                                                                                                                                                        .F10.3,
                                                                                                                                                                                                                                                                                                       11+0+11
                                                                                                                                                                                                                                                                                                                     .F10.3.
                                                                                                                                                                                                                                                                                                                                     ナ・ナリナノ
                                                                                                                                                                                                                                                                                                                                                   F10.3.
                                                                                                                                                                                                                                                                                                                                                                 +++0+/
                                                                                                                                                                                                                                                                                                                                                                                               1/±0±++
                                                                                                                                                                                                                                                                                                                                                                                                                            1/+0+++
                                                                                                                                                                                                                                                                                                                                                                                                                                         ,E13.3)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        #·#O#//
                                                                                                                                                                                                                                                                                                                                                                               ,E13,3,
                                                                                                                                                                                                                                                                                                                                                                                                             ,E13,3,
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     ,E13.3)
                                                                                                                                                                                                                                                           •16·
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     A10=1.
                                                                                                        AINC
                                                                                                                                                                                                           105
                                             2900
                                                                                                                                    3000
                                                                                                                                                                                3030
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      15
```

```
FORMAT(#0±,#TIME= #,E14.7,10X,#SIGNAL+TO+NOISE RATIO= #,E20.7)
7000 CONTINUE
                                                                                                                                     CALL SUMM (I,K,EWL,THETAI,THETA2,THETA3,THETA4,C,ANTE,M,M3)
                                                                                                                                                                                                                                                                       CALL SIGNOS(I+K+D+PHII+W+V+SNR+EWLI+SIGVLT+SV+NV)
                                                                                                                                                 CALL CONG(I+K+D+M3+MM+THETA3+THETA4)
CALL RECTA2(I+K+M3+MM+THETA3+THETA4+V+VCONG)
PRINT 76
DO 6050 N=1+K
PRINT 40+V(N)
                                    CALL PHAGEN (I.PHSHFT, PH, A10, RAN)
DO 6000 N=1, I
                                                                              THETAI (N) = THETAI (N) + RAN (N)
DO 7969 JJ=1,JT.100
DO 6016 JK= 1,100
                                                                                                                                                                                                                                                                                                             PRINT 1501.AJT, SNR
                                                                                                                                                                                                                                                                                           AJT= (99.+JJ) *T
                                                                                                  CONTINUE
                                                                                                                                                                                                                                                       CONTINUE
                                                                                                                   CONTINUE
                                                                                                                                                                                                                                                                                                                                                                     STOP
                                                                                            6000
                                                                                                                                                                                                                                                                                                                                                                  3003
                                                                                                                                                                                                                                                   6050
                                                                                                                                                                                                                                                                                                                            1501
```

```
00020200
                  00020600
                                                                    0002000
                                                                                     00021000
                                                                                                        00021100
                                                                                                                        00021200
                                                                                                                                        00021300
                                                                                                                                                         00021400
                                                                                                                                                                         00021500
                                   0002000
                                                   00020800
                                                                                                                                                                                          00021600
                                                                                                                                                                                                          00021700
                                                                                                                                                                                                                            00021800
                                                                                                                                                                                                                                           00021900
                                                                                                                                                                                                                                                              00022000
                                                                                                                                                                                                                                                                               00022100
                                                                                                                                                                                                                                                                                                00022200
                                                                                                                                                                                                                                                                                                                                                                   00022600
                                                                                                                                                                                                                                                                                                                                                                                      00022700
                                                                                                                                                                                                                                                                                                                                                                                                                                      00023000
                                                                                                                                                                                                                                                                                                                                                                                                                                                                         00023200
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          00023300
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           00023400
                                                                                                                                                                                                                                                                                                                 00022300
                                                                                                                                                                                                                                                                                                                                 00022400
                                                                                                                                                                                                                                                                                                                                                  000225000
                                                                                                                                                                                                                                                                                                                                                                                                      00022800
                                                                                                                                                                                                                                                                                                                                                                                                                       00022900
                                                                                                                                                                                                                                                                                                                                                                                                                                                         00023100
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           00023500
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             00023700
                                                                                                                                        SUBROUTINE SUMM (I.K.EWL.THETAI, THETA2, THETA3, THETA4, C, ANTE.M, M3)
                                                                                                                                                       DOUBLE PRECISION M(I) . EWL (K, I) . THETA! (I) , THETA? (I) , THETA3 (K) ,
                                                                                                                                                                                                                                                                                                                                                                                                                                                     DOUBLE PRECISION M3(K), MM(K), THETA3(K), THETA4(K)
                                                                  EWL (N,NN) = (2,*3,14159*D*DCOS(PHI(NN))) * (N-1)
                 DC05
                                                                                                                                                                                                                                                                                                                                                                                                                                    SUBROUTINE CONG (I.K.D, M3, MM, THETA3, THETA4)
               EWL (K, I), PHI (I)
                                                                                                                                                                                                                                                                                                                                                                                     CALL PHASO2 (I,K,D,THETA3,ANTE,M3)
  SUBROUTINE DISTAN(I,K,D,EWL,PHI)
                                                                                                                                                                                                                                           THETAZ (N) = THETAI (N)-EWL (KI,N)
                                                                                                                                                                                          C(I) , ANTE (K) , ANTEV
                                                                                                                                                                                                                                                                          CALL RECTAL (I.M. THETAZ.C)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        = -THETA3(N)
                                                                                                                                                                                                                                                                                                                                ANTEV = C(J) +ANTEV
                  DOUBLE PRECISION
                                                                                                                                                                                                                                                                                                                                                 ANTE(K1) = ANTEV
                                                                                                                                                                         P THETA4(K) +M3(K)
                                                                                                                                                                                                                                                                                               ANTEV = (0..0.)
                                    00 1 NN = 1.1
                                                                                                                                                                                                          00 2 KI = 19K
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     MM (N) = M3 (N)
                                                                                                                                                                                                                                                                                                                DO 2 J = 1,I
                                                     DO 1 N = 1,K
                                                                                                                                                                                                                                                                                                                                                                                                                                                                        00 1 N = 1.K
                                                                                                                                                                                                                           1.1 = N 1 00
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         THETA4 (N)
                                                                                      CONTINUE
                                                                                                                                                                                                                                                              CONTINUE
                                                                                                                                                                                                                                                                                                                                                                  CONTINUE
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           CONTINUE
                                                                                                                                                                                          COMPLEX
                                                                                                                                                                                                                                                                                                                                                                                                   RETURN
                                                                                                        RETURN
                                                                                                                          END
                                                                                                                                                                                                                                                                                                                                                                                                                       END
```

```
00023900
                                   00024000
00023800
                                                                                         00024300
                                                                                                                                              00024600
                                                                                                                                                                00024700
                                                                                                                                                                                                  00024900
                                                                                                                                                                                                                    00022000
                                                                                                                                                                                                                                                      00025200
                                                      00024100
                                                                        00024200
                                                                                                           00024400
                                                                                                                            00024500
                                                                                                                                                                                 00024800
                                                                                                                                                                                                                                     00025100
                                                                                                                                                                                                                                                                        00025300
                                                                                                                                                                                                                                                                                          00025400
                                                                                                                                                                                                                                                                                                           00025500
                                                                                                                                                                                                                                                                                                                              00025600
                                                                                                                                                                                                                                                                                                                                              00025700
                                                                                                                                                                                                                                                                                                                                                                00025800
                                                                                                                                                                                                                                                                                                                                                                                 00025900
                                                                                                                                                                                                                                                                                                                                                                                                   00026000
                                                                                                                                                                                                                                                                                                                                                                                                                     00026100
                                                                                                                                                                                                                                                                                                                                                                                                                                       000292000
                                                                                                                                                                                                                                                                                                                                                                                                                                                         00026300
                                                                                                                                                                                                                                                                                                                                                                                                                                                                           00026400
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           00026500
               DOUBLE PRECISION M3 (K) , THETA3 (K) , THETA4 (K) , MM (K) , A, B, A1, B1, DCOS,
                                                                                                                                                                                                                                                                                        DOUBLE PRECISION M3(K), THETA3(K), OCOS, DSIN, DREAL, DMAG
SUBROUTINE RECTAZ(1,K,M3,MM,THETA3,THETA4,V,VCONG)
                                                                                                                                                                                                                                                                                                                                                           DREAL = (ANTE(N) + CONJG(ANTE(N)))/2.

DMAG = ((ANTE(N) - CONJG(ANTE(N)))/2.) * (0.,-1.)
                                                                                                                                                                                                                                                                       SURROUTINE PHASO2(I+K+D+THETA3, ANTE, M3)
                                                   COMPLEX V(K) . VCONG(K) , ANTEV , CMPLX
                                                                                                                                                                                                                                                                                                                             ANTE(K), CMPLX, CONJG
                                                                                                                                                                                                                                                                                                                                                                                                  M3(N) = DSORT (DREAL **2+DMAG**2)
                                                                                                                                                                                                                                                                                                         DOUBLE PPECISION DSORT, DATAN
                                                                                                                                                                                                                                                                                                                                                                                                                  THETA3(N) = DATAN(DMAG/DREAL)
                                                                                                                                                         = MM(N) *DCOS (THETA4 (N))
                                                                                                      A = M3(N) *DCOS(THETA3(N))
                                                                                                                                                                                                  = DCMPLX(A1,81)
                                                                                                                                              = DCMPLX(A+B)
                                                                       COMPLEX*16 DCMPLX
                                                                                                                                                                                                                                                                                                                                                                                                                                                          M3(N)= -M3(N)
                                                                                          DO 1 N = 1.K
                                                                                                                                                                                                                                                                                                                                               DO 1 N = 1.K
                                                                                                                                                                                                                                                                                                                                                                                                                                        IF IDREAL
                                                                                                                                                                                                    VCONG (N)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                           CONTINUE
                                                                                                                                                                                                                    CONTINUE
                                                                                                                                                                                                                                                                                                                              COMPLEX
                                                                                                                                                                                                                                       RETURN
                                     NISO .
                                                                                                                                          (N)
                                                                                                                                                                                                                                                        END
```

```
00026700
                            00026900
                                           00027000
                                                         00027100
               00026800
                                                                       00027200
                                                                                      00027300
                                                                                                    00027400
                                                                                                                  00027500
                                                                                                                                 00027600
                                                                                                                                               00027700
                                                                                                                                                             00027800
                                                                                                                                                                           00027900
                                                                                                                                                                                         00028000
                                                                                                                                                                                                      00028100
                                                                                                                                                                                                                     00028200
                                                                                                                                                                                                                                    00028300
                                                                                                                                                                                                                                                                00028500
                                                                                                                                                                                                                                                                             00028600
                                                                                                                                                                                                                                                                                             00028700
                                                                                                                                                                                                                                                                                                          00028800
                                                                                                                                                                                                                                                                                                                        00028900
                                                                                                                                                                                                                                                                                                                                      00052000
                                                                                                                                                                                                                                                                                                                                                    00029100
                                                                                                                                                                                                                                                                                                                                                                   00029200
                                                                                                                                                                                                                                                                                                                                                                                 00029300
                                                                                                                                                                                                                                                                                                                                                                                                00056700
                                                                                                                                                                                                                                                                                                                                                                                                              000562000
                                                                                                                                                                                                                                                                                                                                                                                                                           00053600
                                                                                                                                                                                                                                                                                                                                                                                                                                         00029700
                                                                                                                                                                                                                                                  00028400
             DOUBLE PRECISION M(I), THETA2(I), DCOS, DSIN, A, B
SUBROUTINE RECTAL (I,M,THETAZ,C)
                                                                                                                                                                                                                                                                                                                                                                                SUBROUTINE MULTI(K.SUM.VCONG,Z)
                            C(1), ANTEV, CMPLX
                                                                                                                                                            SUBROUTINE WEIGHT (K, W.V, WV)
                                                                                                                                                                                                                                                                                                                                                                                              COMPLEX VCONG(K) . Z(K) . SUM
                                                                    A = M(N) *DCOS (THETAZ (N))
                                                                                    B = M(N) *DSIN(THETA2(N))

C(N) = DCMPLX(A,B)
                                                                                                                                                                                                                                                              SUBROUTINE ADD (K, WV, SUM)
                                                                                                                                                                           COMPLEX W(K) . V(K) . WV(K)
                                                                                                                                                                                                                                                                                                                                                                                                                         Z(I) = VCONG(I) *SUM
                                          COMPLEX*16 DCMPLX
                                                                                                                                                                                                                                                                             COMPLEX WV (K) . SUM
                                                                                                                                                                                                      (I) M*(I) = (I) M*(I)
                                                                                                                                                                                                                                                                                                                       SUM = WV(I)+SUM
                                                                                                                                                                                                                                                                                                        DO 1 I = 1.K
                                                                                                                                                                                        DO 1 I = 1,K
                                                                                                                                                                                                                                                                                                                                                                                                              I = 1.K
                                                         1,1 = N 1 00
                                                                                                                                                                                                                                                                                            SUM=(6.00.)
                                                                                                                                                                                                                                                                                                                                     CONTINUE
                                                                                                                 CONTINUE
                                                                                                                                                                                                                    CONTINUE
                                                                                                                                                                                                                                                                                                                                                                                                                                         CONTINUE
                            COMPLEX
                                                                                                                                 RETURN
                                                                                                                                                                                                                                   RETURN
                                                                                                                                                                                                                                                                                                                                                                                                                                                       RETURN
                                                                                                                                                                                                                                                                                                                                                    RETURN
                                                                                                                                                                                                                                                                                                                                                                                                            00
                                                                                                                                               END
                                                                                                                                                                                                                                                END
                                                                                                                                                                                                                                                                                                                                                                   END
```

```
00102000
                               00030500
                                                                                                              00030760
                                                                                                                              00030800
                                                                                                                                             00602000
                                                                                                                                                            00031000
                                                                                                                                                                                                                           00031400
                                                                                                                                                                                                                                          00031500
                                                                                                                                                                                                                                                           00031600
                                                                                                                                                                                                                                                                                       00031800
                                                                                                                                                                                                                                                                                                                         00032000
                                                                                                                                                                                                                                                                                                                                                                                                                                     00032700
                                                                                                                                                                                                                                                                                                                                                                                                                                                      00032800
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    00022000
00000000
                                              00030300
                                                               00030400
                                                                              00030200
                                                                                              00906000
                                                                                                                                                                             00031100
                                                                                                                                                                                           00031200
                                                                                                                                                                                                           00031300
                                                                                                                                                                                                                                                                         00031700
                                                                                                                                                                                                                                                                                                        00031900
                                                                                                                                                                                                                                                                                                                                         00032100
                                                                                                                                                                                                                                                                                                                                                        00032200
                                                                                                                                                                                                                                                                                                                                                                       00032300
                                                                                                                                                                                                                                                                                                                                                                                      00032400
                                                                                                                                                                                                                                                                                                                                                                                                                      00032600
                                                                                                                                                                                                                                                                                                                                                                                                                                                                     00032900
                                                                                                                                                                                                                                                                                                                                                                                                      00032500
                                                                                                                                                                                            PARTI (N) = (1-T/TAU) *PARTI (N) + (T/TAU) *PART2 (N)
                                                                                                                                             SUBROUTINE LPF (K. I. TAU. PARTI, PARTZ, NPNP)
                                                                                                                                                                                                                                                            SUBROUTINE ADD2 (K, PARTI, PART2, A, G) DIMENSION PARTI(K), PART2(K), A(K)
                                                                                                                                                                                                                                                                                                                                                                       SUBROUTINE TOGETH(K, PARTI, PARTZ, A)
  SUBROUTINE BRAKE (K, A, PART), PART2)
                                                                                                                                                                                                                                                                                                                                                                                                       COMPLEX A(K). CMPLX
DO 1 N = 1.K
A(N) = CMPLX(PART1(N). PART2(N))
               DIMENSION PARTI(K) . PARTZ(K)
                                                                                                                                                            DIMENSION PARTI (K) . PARTZ (K)
                                                                                                                                                                                                                                                                                                        A(I) = (PART1 (I) -PART2 (I)) *G
                                                                                                                                                                                                                                                                                                                                                                                      DIMENSION PARTI (K) , PARTZ (K)
                                                                               = AIMAG(A(I))
                                                              = REAL (A(I))
                                                                                                                                                                               00 1 N = 1.K
                                                 00 1 I = 1.K
                                                                                                                                                                                                                                                                                           00 1 I = 1 • K
                                  COMPLEX A(K)
                                                                               PARTZ(I)
                                                                                                                                                                                                                                                                                                                          CONTINUE
                                                                                                                                                                                                             CONTINUE
                                                                 PART) (I)
                                                                                                CONTINUE
                                                                                                                                                                                                                                                                                                                                                                                                                                                       CONTINUE
                                                                                                                                                                                                                            RETURN
                                                                                                               RETURN
                                                                                                                                                                                                                                                                                                                                          RETURN
                                                                                                                                                                                                                                                                                                                                                                                                                                                                       RETURN
                                                                                                                                                                                                                                             END
```

```
COMPLEX W(K), SIGVLT(K), CMPLX, V(K), SV(K), NV(K), ANOSUM, SIGSUM
SUBROUTINE SIGNOS(I,K,D,PHII,W,V,SNR,EWLI,SIGVLT,SV,NV)
DOUBLE PRECISION EWLI(K,1),PHII(1),DCOS,DSIN,AAA,BBB
                                                                                             THETA8(1), THETA9(1), MM1(1), MM2(1)
                                                                                                                                                                                          PHASO2 (1.1.0,THETA8, ANDSUM, MM1)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                         PHASO2 (I+1+0+THETA9+SIGSUM+MM2)
                                                                                                                     DOUBLE PRECISION ANOSOR, SIGSOR, SNR
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               SNR= DLOGIO (SIGSOR/ANOSOR) *10.
                                                                                                                                                                                                                                                          FORMAT (#0#+#MM] (1) = #+2E20.7)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 FORMAT (#0#, #MM2(1) = #, 2E20.7)
                                                                                                                                                                                                                                                                                  CALL DISTAN(1,K,D,EWL1,PHI1)
                                                                                                                                                                                                                                                                                                                                                                                                                           CALL WEIGHT (K+W+SIGVLT+SV)
                                                                                                                                                                                                                                                                                                                                                                            SIGULT (N) =DCMPLX (AAA + BBB)
                                                                                                                                          CALL WEIGHT (K, W.V. NV)
                                                                                                                                                                    ADD (K.NV. ANOSUM)
                                                                                                                                                                                                                                                                                                                                                                                                                                                    ADD (K.SV. SIGSUM)
                                                                                                                                                                                                                                                                                                                                    AAA= DCOS(EWL1(N+1))
                                                                                                                                                                                                                                                                                                                                                        BBB=-DSIN(EWL1(N.1))
                                                                                                                                                                                                                  ANOSOR = MM1 (1) **2
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              SIGSOR = MM2(1) **2
                                                                          COMPLEX*16 DCMPLX
                                                                                                                                                                                                                                       PRINT 10.MM1(1)
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               PRINT 40. MMZ(1)
                                                                                                                                                                                                                                                                                                            N= 1.K
                                                                                                                                                                                                                                                                                                                                                                                                        CONTINUE
                                                                                                  REAL *8
                                                                                                                                                                     CALL
                                                                                                                                                                                                                                                                                                                                                                                                                                                    CALL
                                                                                                                                                                                                                                                                                                            00
                                                                                                                                                                                                                                                                                                                                                                                                                                                                           CALL
                                                                                                                                                                                            CALL
                                                                                                                                                                                                                                                              10
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       40
```

```
RAN(N) = (2.*RAN(N) *PHSRNG* (PI/180.) J/AITMC
SUBROUTINE PHAGEN(I, PHSRNG, PH, AITMC, RAN)
DIMENSION RAN(I)
PI= 3.14159
                                                                                                                                                                                               PH= RAN(1)*100.

DO 2000 N= 1.1

IF(RAN(N) .LT. 0.5) GO TO500

RAN(N) = -(RAN(N) - 0.5)

A= RAN(N)
                                                                                   RAN(1) = ANN-AN

IF (I .EQ. 1) GO TO 100

DO 100 N=2.1

AN= (RAN(N-1) + PI)**8.
                                                                                                                                                                       RAN(N) = AN- BN
                                                                                                                                                                                    CONTINUE
                                                                                                                                                                                                                                                                                      CONTINUE
                                                          NN= NN
                                            ANN=PH
                                                                        AN= NN
                                                                                                                                            IN= AN
                                                                                                                                                         BN= IN
                                                                                                                                                                                                                                                                                                  RETURN
                                                                                                                                                                                                                                                                                   2000
                                                                                                                                                                                                                                                         200
                                                                                                                                                                                 100
```

```
00034000
                                                                                                                                       00034200
 00033100
          00033200
                             00033400
                                       00033500
                                                00033600
                                                           00033700
                                                                    00033800
                                                                             00033900
                                                                                                 00034100
                                                                                                          00034200
                                                                                                                    00034300
                                                                                                                              00034400
                                                                                                                                                 00034600
                                                                                                                                                          00034700
                                                                                                                                                                                                 00035100
                                                                                                                                                                                                          00035200
                                                                                                                                                                                                                  00035300
                                                                                                                                                                     00034800
                                                                                                                                                                              00034900
                                                                                                                                                                                       00035000
                                                                                                                                                                                                                                                                                                 .05 E-03
                                                                                                                                                                                                                                                                                                                                       25.
SUBROUTINE CHECK (K+W+WM1+WW2+W1+W2,666)
DIMENSION W1(K)+W2(K)+WW1(K)+WW2(K)
                                                                                                                                                                                                                                                                                                .01 E-03
                                                                                                                    6+1
                                                                                                           = 6+1
                                                                                                           IF (ABS (CHECK2) .GT..00005) G = CONTINIF
                                                                                                                                                                                                                                                                                                                                       -25.
                                                                                                                                                                                                                                                                                                                              105.
                                                                                       CHECKI = WWI (J)-WI (J)
                                                                                                   CHECK2 = WW2(J)-W2(J)
                                                                                                                              CONTINUE
IF (G.LT..5) GO TO 14
DO 3 J = 1.K
                                                            WZ(J) = AIMAG(W(J))
                                                 WI(J) = REAL (W(J))
                                                                                                                                                                                                                                                                                              0.2E-05
0.2E-05
0.375E02
                                                                                                                                                            = W1(3)
                                                                                                                                                                      W2(J)
                                                                               00 2 J = 1.K
                                        DO 1 J = 1.K
                      COMPLEX W(K)
                                                                                                                                                                                                                                      * 00 NISAS 109//
                                                                      CONTINUE
                                                                                                                                                                      WW2(J) =
                                                                                                                                                                                                 CONTINUE
                                                                                                                                                                               CONTINUE
                                                                                                                                                                                                            666 = 1.
                                                                                                                                                                                                                                                                                        135
                                                                                                                                                                                                                    RETURN
                                                                                                                                                                                        RETURN
                                6 = 0.
                                                                                                                                                           WWI (J)
                                                                                                                                                                                                                                                           18
                                                                                                                                                                                                   14
                                                                                                                                2
                                                                                                                                                                                                                                                                            10.
                                                                                                                                                                                                                                                                                                                                                            1
```

## REFERENCES

- [1] S. P. Applebaum, "Adaptive arrays," Syracuse University Research Copr., Rept. SPL TR 66-1, August 1966.
- [2] B. Widrow et al., "Adaptive antenna systems," Proceedings of the IEEE Vol. 55, pp. 2143-2159, December 1967.
- [3] L. E. Brennan, E. L. Pugh, I. S. Reed, "Control loop noise in adaptive array antennas," <u>IEEE Trans. Aerospace and Electronic Systems</u>, Vol. AES-7, pp. 254-262, March 1971.
- [4] De Russo, Roy, Close State Variables for Engineers, John Wiley & Sons, Inc., New York, 1965.
- [5] Le. E. Brennan, I. S. Reed, "Theory of adaptive radar," <u>IEEE Trans.on Aerospace and Electronic Systems</u>, Vol. AES-9, pp. 237-252, March 1973.
- [6] J. B. Thomas, An Introduction to Statical Communication Theory, John Wiley & Sons, Inc., New York, 1969.
- [7] M. I. Skolnik, Introduction to Radar Systems, McGraw-Hill Book Co., New York, 1962.
- [8] B. Widrow, "Adaptive filters I: Fundamentals," Stanford University Report SU-SEL-66-126, December, 1966.
- [9] J. A. Stratton, Electromagnetic Theory, McGraw-Hill Book Company, New York, 1941.
- [10] L. E. Brennan, "Adaptive antenna processing and MTI techniques,"
  Notes from a short course, Technology Service Corporation, March 1973.
- [11] IEEE Trans. Antennas Propagation. (Special issue on Active and Adaptive Antennas), vol. AP-12, March 1964.
- [12] D. L. Margerum, "Self-phased arrays," in <u>Microwave Scanning Antennas</u>, Vol. 3, R. C. Hansen, Ed. New York: Academic Press, 1966, Chapter 5.
- [13] S. W. W. Shor, "Adaptive technique to discriminate against coherent noise in a narrow-band system," J. Acoust. Soc. Amer. Vol. 39, pp. 74-78, January 1966.

- [14] L. J. Griffiths, "A simple adaptive algorithm for real-time processing in antenna systems," Proc. IEEE, Vol. 57, pp. 1696-1704, October 1969.
- [15] O. L. Frost, III, "An algorithm for linearly constrained adaptive Array Processing," Proc. IEEE, Vol. 60, pp. 926-935, August 1972.
- [16] R. L. Riegler and R. T. Compton, Jr., "An adaptive array for interference rejection," Proc. IEEE, Vol. 61, pp. 748-758, June 1973.
- [17] C. L. Zahm, "Application of adaptive arrays to supress strong jammers in the presence of weak signals," <u>IEEE Trans. Aerosp. Electron.</u> <u>System</u>, Vol. AES-9, pp. 260-271, March 1973.
- [18] L. E. Brennan and I. S. Reed, "Effect of envelope limiting in adaptive array control loops," <u>IEEE Trans. Aerosp. Electron, Syst.</u>, Vol. AES-7, pp. 698-700, July 1971.
- [19] I. S. Reed, J. D. Mallett, and L. E. Brennan, "Rapid convergence rate in adaptive arrays," <u>IEEE Trans. Aerosp. Electron, Syst.</u>, Vol. AES-10, pp. 853-863, November 1974.
- [20] L. Stark, "Microwave theory of phased array antennas-A review," Proc. IEEE, Vol. 62, pp. 1661-1701, Dec. 1974.
- [21] F. B. Hildebrand, Methods of Applied Mathematics, 2nd ed. Englewood Cliffs, N.J.: Prentice-Hall, 1965.
- [22] R. F. Harrington, Field Computation by Moment Methods, New York: Macmillian, 1968.
- [23] D. E. N. Davies, "Independent angular steering of each zero of the directional pattern for a linear array," <u>IEEE Trans. Antennas</u> Propagat. (Communication), Vol. AP-15, pp. 296-298, March 1967.
- [24] W. F. Gabriel, "Adaptive arrays-An introduction," Proc. IEEE, Vol. 64, pp. 239-272, February 1976.
- [25] P. W. Howells, "Intermediate frequency side-lobe canceller," U.S. Patent 3,202,990, August 24, 1965 (filed May 4, 1959).